А.В. Строгонов Н.Н. Кошелева
А.Б. Буслаев

ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ В БАЗИСЕ ПЛИС: ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Учебное пособие

Воронеж 2017

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО «Воронежский государственный
технический университет»

А.В. Строгонов Н.Н. Кошелева
А.Б. Буслаев

ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ В БАЗИСЕ ПЛИС: ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Утверждено учебно-методическим советом
университета в качестве учебного пособия

Воронеж 2017

УДК 621.3.049.77.001.63(075.8)
ББК 32.97-018.2я7
С 861

Строгонов А.В. Проектирование цифровых устройств в базисе ПЛИС: лабораторный практикум: учеб. пособие [Электронный ресурс]. – Электрон. текстовые и граф. данные (3,7 Мб) / А.В. Строгонов, Н.Н. Кошелева, А.Б. Буслаев. - Воронеж: ФГБОУ ВО «Воронежский государственный технический университет», 2017. – 1 электрон. опт. диск (CD-ROM): цв. – Систем. требования: ПК 500 и выше; 256 Мб ОЗУ; Windows XP; SVGA с разрешением 1024x768; Adobe Acrobat; CD-ROM дисковод; мышь. – Загл. с экрана.

В учебном пособии рассматривается проектирование комбинационных и последовательностных устройств с использованием САПР программируемых логических интегральных схем (ПЛИС) Xilinx ISE.

Издание соответствует требованиям Федерального государственного образовательного стандарта высшего образования по направлению подготовки бакалавров 11.03.04 «Электроника и наноэлектроника», направленности «Микроэлектроника и твердотельная электроника», дисциплинам «Проектирование ПЛИС», «Проектирование цифровых устройств в базисе ПЛИС».

Табл. 22. Ил. 48. Библиогр.: 3 назв.

Рецензенты: кафедра физики полупроводников и микроэлектроники Воронежского государственного университета (зав. кафедрой д-р физ.-мат. наук, проф. Е.Н. Бормонтов);
д-р техн. наук, проф. М.И. Горлов

ВВЕДЕНИЕ

Программируемые логические интегральные схемы (ПЛИС) все более широко используются для создания цифровых систем различного назначения. Фирма Xilinx, являясь ведущим мировым производителем ПЛИС, предоставляет разработчикам широкий спектр кристаллов с различной технологией производства, степенью интеграции, архитектурой, быстродействием, потребляемой мощностью и напряжением питания.

Интегрированная система автоматизированного проектирования Integrated Software Environment (САПР ISE) является основным продуктом сквозного проектирования цифровых систем на базе ПЛИС фирмы Xilinx. САПР, поддерживает все выпускаемые ПЛИС.

В учебном пособии излагаются основные сведения о порядке работы с версией пакета ISE Webpack.

Цель учебного пособия – на основе примера проекта простейшего логического устройства продемонстрировать основные процедуры проектирования, чтобы максимально сократить промежуток времени между первым знакомством со средой проектирования ISE и конфигурированием проекта своего оригинального устройства в ПЛИС.

Издание соответствует требованиям Федерального государственного образовательного стандарта высшего образования по направлению подготовки бакалавров 11.03.04 «Электроника и наноэлектроника», направленности «Микроэлектроника и твердотельная электроника», дисциплинам «Проектирование ПЛИС», «Проектирование цифровых устройств в базисе ПЛИС».

1. ОПИСАНИЕ РАБОТЫ В САПР XILINXISE

1.1. Создание проекта в САПР XilinxISE

Проектом в САПР ISE является совокупность файлов, которые содержат информацию, необходимую и достаточную для выполнения всех этапов разработки цифрового устройства. При разработке цифровых устройств на базе ПЛИС Xilinx условно можно выделить следующие основные этапы проектирования:

- анализ задачи, разработка алгоритма работы устройства, разбиение проекта на модули, определение семейства ПЛИС, типа кристалла, корпуса, а также средств синтеза;
- разработка описания проектируемого устройства и его отдельных модулей в форме принципиальной схемы, кода поведенческого описания на языке HLD (Hardware Language Description);
- синтез модулей и всего устройства;
- функциональное моделирование;
- размещение и трассировка проекта в кристалле;
- оптимизация устройства по временным характеристикам, потребляемой мощности и ресурсам ПЛИС;
- загрузка проекта в кристалл (программирование ПЛИС).

Для создания проекта нового устройства следует запустить на исполнение основную программу среды, откроем САПР XilinxISE.

В результате откроется начальное окно работы с программой (рис. 1.1).

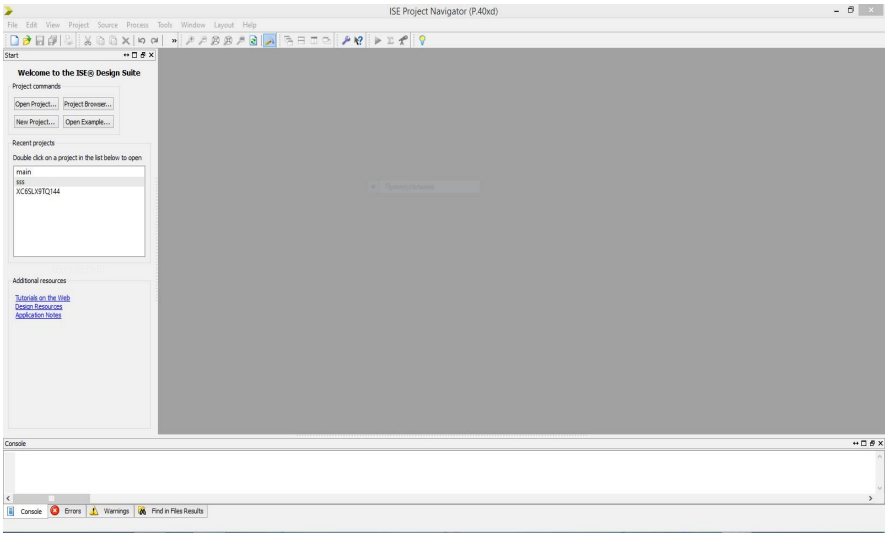


Рис. 1.1. Окно ProjectNavigator САПР

Создадим новый проект, выбираем в меню File/NewProject. Откроется окно настроек создания нового проекта «NewProjectWizard». В первом окне настроек проекта (рис. 1.2) вводятся имя(name), локация(location), рабочая папка проекта (workingdirectory), краткое описание и указывается формат файла верхнего уровня иерархии (top-levelsourcetype). В нашем случае указываем имя проекта: LR_1, локация и рабочая папка одинаковы: D:\Xproject\LR_1. Для простоты поиска проекта разрешается указать краткое описание проекта и личные данные. Файл верхнего уровня иерархии указываем «HDL». Жмем «Next».

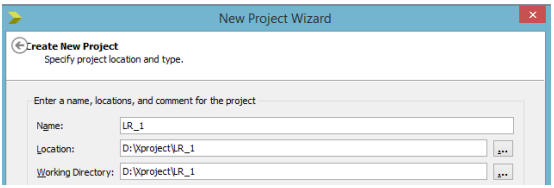


Рис. 1.2. Окно «NewProjectWizard» с настройками расположения и имени проекта

Во втором окне настроек указываются спецификации на используемую ПЛИС и настройки используемого синтезатора, симулятора, также указываются настройки используемого языка и его версии. Выбираем:используемая отладочная плата (EvaluationDevelopmentboard) – «Nonespecified», категория продукта (Productcategory) – «ALL», семейство ПЛИС (family) – «Spartan 6», устройство (device) – «XC6SLX9», тип корпуса (package) – «TQG144», скоростной показатель (speed) - «-2». Приведенные выше настройки соответствуют плате фирмы LDM-systemsXB-XC6SLX9 TQG144 – evolution. При использовании иной ПЛИС необходимо вводить соответствующие настройки.

Программные настройки: устройство синтеза (synthesistool) – «XST», симулятор (simulator) – «iSim», предпочтительный язык (preferredlanguage) – «VHDL» (данная настройка является формальной, независимо от указанного языка в САПР XilinxISE сохраняется возможность совместного использования VHDL + Verilog в одном проекте), стандарт языка VHDL – «VHDL-93», остальные настройки без изменений. Окно с необходимыми настройками приведено на рис. 1.3.

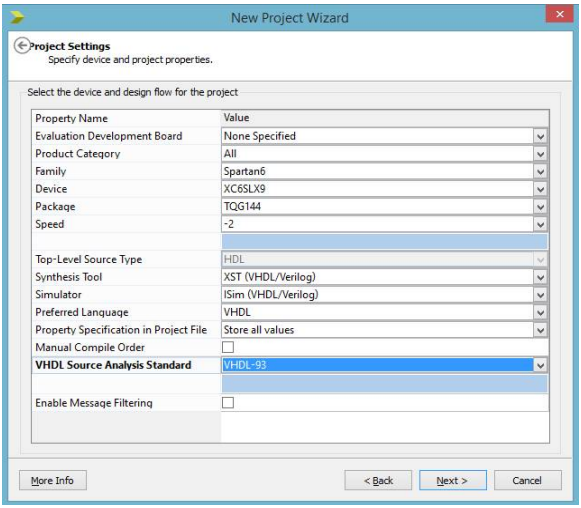


Рис. 1.3. Окно «NewProjectWizard» с настройками используемой ПЛИС

Третье окно суммарной проверки настроек создаваемого проекта (рис. 1.4). Проверяем настройки при нахождении отклонений от требуемых возвращаемся назад, кликнув по стрелочке в верхнем левом углу окна. При верности настроем жмем «Finish».

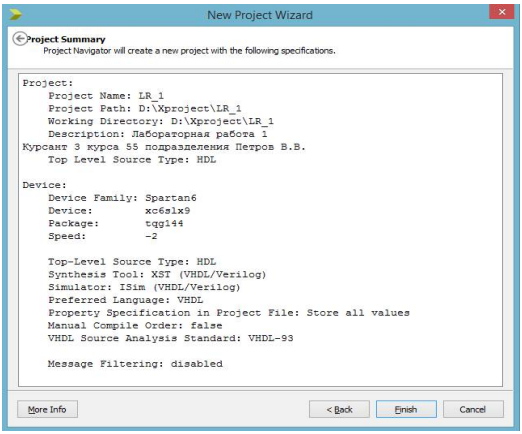


Рис. 1.4. Окно «NewProjectWizard» суммарной проверки настроек

1.2. Создание нового файла исходного кода

Новый пустой проект создан (рис. 1.5). Добавим новый файл исходного кода, выбираем меню project/newsource. Откроется окно настроек нового файла исходного кода (рис. 1.6). В настройках выбираем «VHDL module», указываем имя модуля «vhdl_or», локация та же что и у проекта D:\Xproject\LR_1 и ставим галочку напротив «addtoproject» (добавить в проект). Жмем «Next». Следующее окно портов создаваемого модуля (рис. 1.7), окно является формальным, так как только добавляет в исходный код описание портов модуля, в любом случае введенные порты можно исправить в исходном коде. Пропускаем и жмем «Next». Последнее окно проверки настроек (рис. 1.8). Проверяем, если необходимо возвращаемся и исправляем. Убедившись, что все верно создаем файл, нажав «Finish».

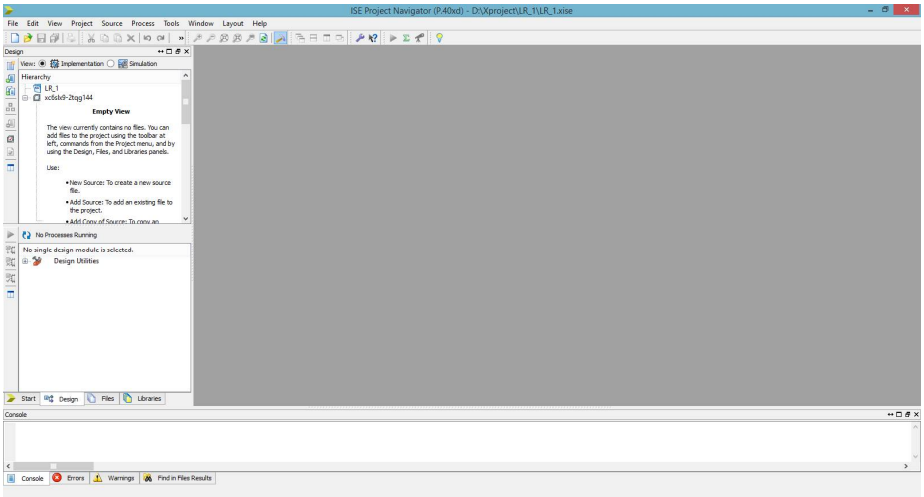


Рис. 1.5. Окно ProjectNavigator САПР XilinxISE
с пустым проектом

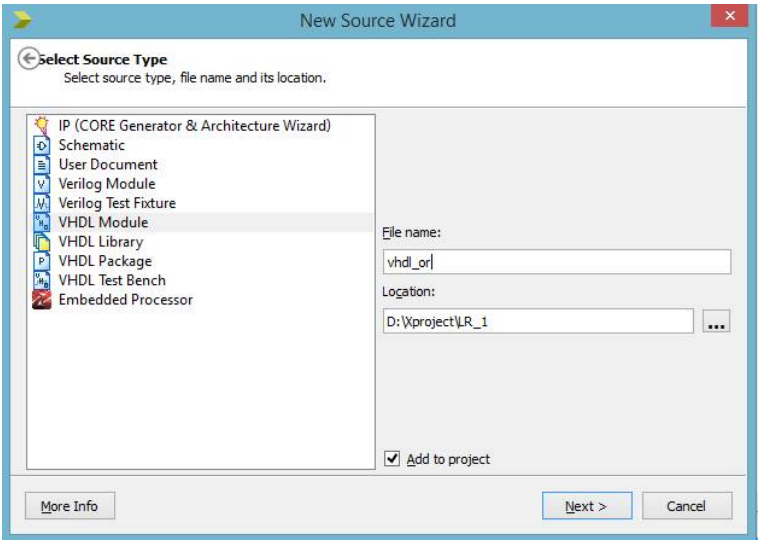


Рис. 1.6. Окно «NewSourceWizard» настроек расположения
и типа нового файла исходного кода

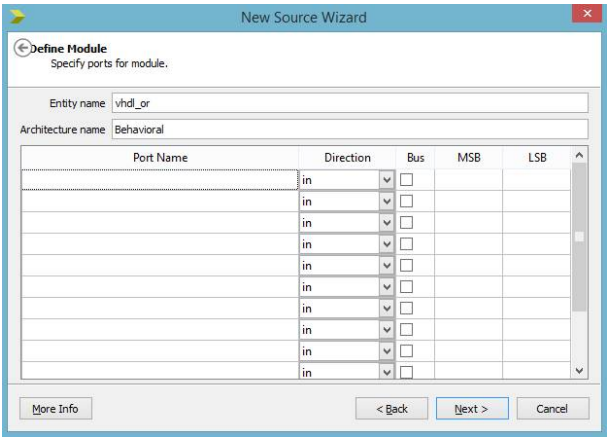


Рис. 1.7. Окно «NewSourceWizard» настроек портов модуля

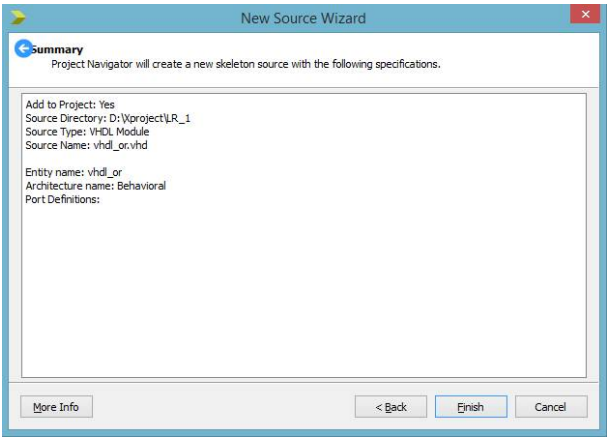


Рис. 1.8. Окно «NewSourceWizard» проверки настроек

Новый файл исходного кода добавлен (рис. 1.9). Установим новый файл как верхний файл иерархии, выбираем во вкладке «Design» необходимый файл и в меню source/setastopmodule. О том, что файл является верхним, в иерархии свидетельствует иконка из трех квадратов рядом с именем файла. Так как файл в проекте всего один, то САПР ав-

томатически устанавливает его верхним. При синтезации проекта (в HDL языках код не компилируется а синтезируется в RTL описание) просинтезируются только файл верхнего уровня и все входящие в него модули. Один и тот же модуль может всходить в несколько файлов. Добавим в новый файл описание логического элемента «ИЛИ» (рис. 1.10). Исходный код элемента приведен ниже.

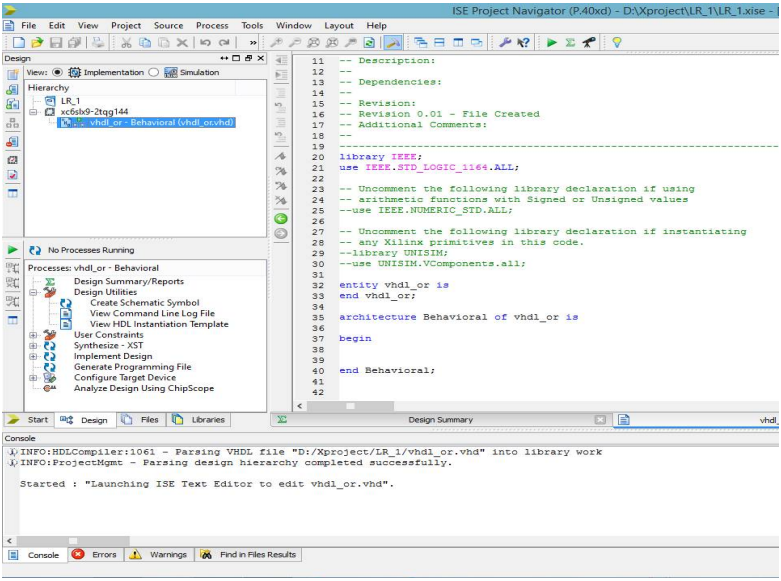


Рис. 1.9. Окно ProjectNavigator САПР XilinxISE
с новым файлом исходного кода

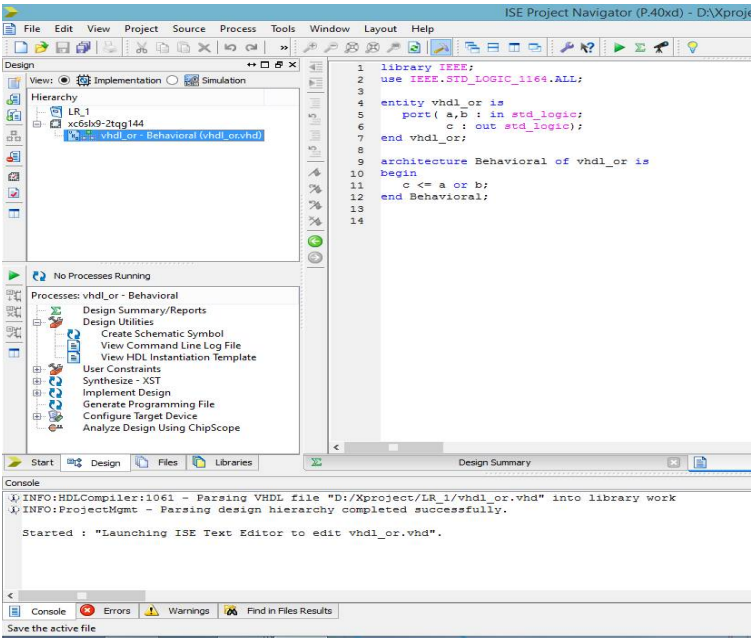


Рис. 1.10. Окно ProjectNavigator САПР XilinxISE с описанием логического элемента «ИЛИ»

Листинг кода на языке VHDL: описание логического элемента «ИЛИ»

libraryIEEE;	базовый пакет библиотек;
useIEEE.STD_LOGIC_1164.ALL;	библиотека стандартных логических элементов;
entity vhdl_oris	начало описания объекта vhdl_or;
port(a,b : in std_logic; c : out std_logic);	описание портов;
end vhdl_or;	конец описания объекта vhdl_or;
architecture Behavioral of vhdl_or is	начало описания архитектуры vhdl_or;

begin	начало описания алгоритма работы;
c <= a or b;	алгоритм
end Behavioral;	конец описания архитектуры vhdl_or

После ввода описания логического элемента сохраним - проверим исходный код на ошибки. В том случае если файл не является верхним, то дважды кликаем по «CheckSyntax»в меню процессов во вкладке «Design» (рис. 1.11). Проверка файла верхнего уровня иерархии расположена в том же меню процессов, но в подменю «SynthesizeXST» (рис. 1.12). После проведения, какого либо процесса в меню процессов выводится статус. Всего статусов три: без ошибок (пример рис. 1.12, галочка в зеленом круге), критическая ошибка (пример рис. 1.11, крест в красном круге), незначительная не нарушающая работу ошибка (восклицательный знак на желтом треугольнике).

Данный процесс «Checksyntax» только проверяет синтаксис исходного кода на наличие ошибок в языке, этот процесс не распространяется на подмодули содержащиеся в верхнем файле, также данный процесс не свидетельствует о синтезируемости конструкции. Если проверка выявит ошибки то в меню процессов покажется статус ошибки, в консоли выведется сообщение «checksyntaxfailed», место ошибки возможно подсветить, найдя в консоли сообщение об ошибке и нажав на подсвеченный гипертекст. Желтый треугольник подсветит расположение ошибки в редакторе (рис. 1.13). В данном случае в сообщении указано «near “end”» - перед словом end не хватает двоеточия с запятой. Исправляем, повторяем проверку, убедившись, что синтаксис верный переходим к этапу синтеза.

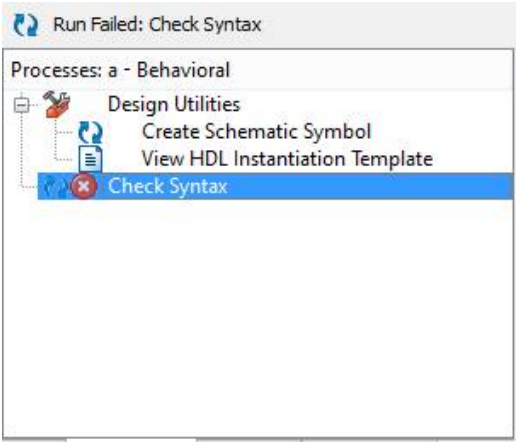


Рис. 1.11. Вид меню процессов для простого файла проекта

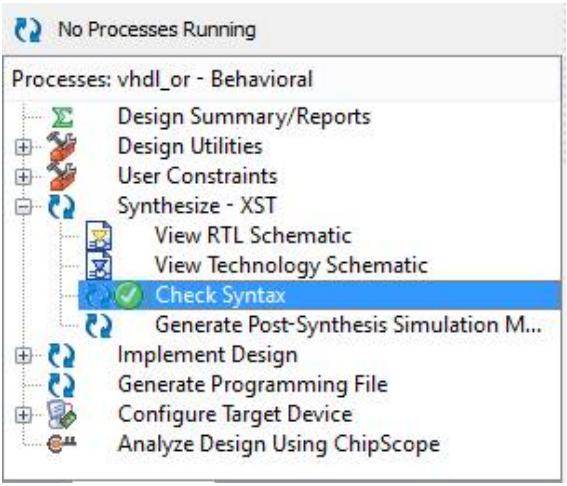


Рис. 1.12. Вид меню процессов для простого файла верхнего уровня иерархии

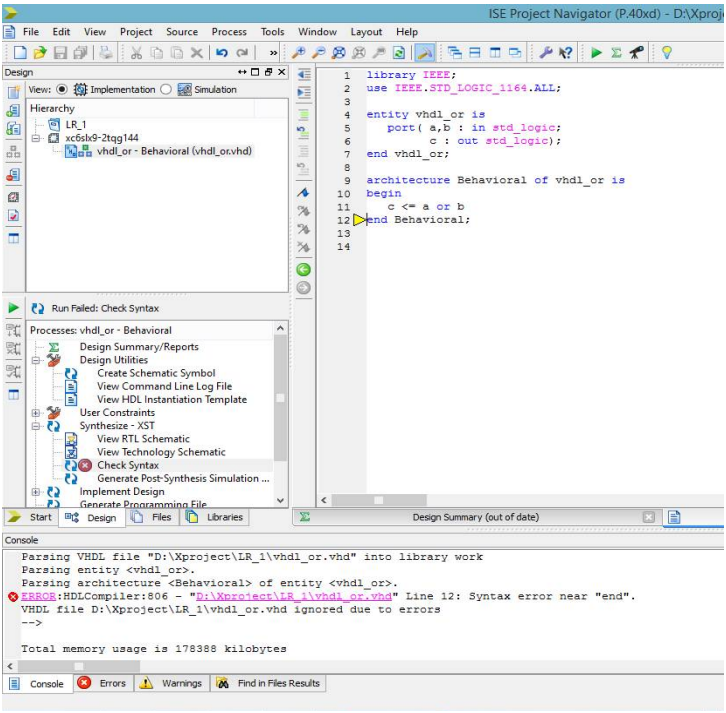


Рис. 1.13. Окно Project Navigator САПР Xilinx ISE с выводом ошибки проверки синтаксиса

1.3. Синтез на основе описания RTL схемы

Синтез логической схемы представляет собой перевод описания на языке HDL в RTL описание представляющее схему как набор логических элементов и межсоединений. Синтез логической схемы в САПР XilinxISE всегда выполняется только для файла верхнего уровня иерархии и входящих в его состав модулей. Для синтеза схемы выберем в меню процессов команду «SynthesizeXST». По мере выполнения операции синтеза в консоли выводится отчет о работе каждой подпрограммы. При выполнении операции синтеза в консоли выводится сообщение «Process "Synthesize - XST" completedsuccessfully» (рис. 1.14). В случае неправильности проекта на каком либо этапе синтез остановится и в консоль выводится сообщение «Process

"Synthesize - XST" completedfailed». При каких либо изменениях в проекте операция синтеза повторяется вновь. Для просмотра результата необходимы в меню процессов выбрать операцию просмотра RTL схемы расположенной в SynthesizeXST/viewRTLSchematic. При готовности RTL схемы САПР XilinxISE запросит вид вывода схемы: в виде блока начиная с верхнего уровня иерархии (top-levelblock) или в виде поэлементного просмотра(ExplorerWizard). Выбираем с верхнего уровня и жмем «ОК» (рис. 1.15). При просмотре RTL схем двойной клик по элементу раскрывает его, если есть возможность (рис. 1.16). Аналогично есть возможность просмотреть синтезируемую схему в виде стандартных элементов ПЛИС, в том виде как полученная схема будет реализована в ПЛИС, команда SynthesizeXST/viewTechnologySchematic (рис. 1.17). На выводе схемы видна одна LUT ячейка (LookUpTable – досл. таблица перекодировки, таблица истинности, стандартный элемент ПЛИС для реализации любых комбинационных выражений, большие логические выражения разбиваются на более маленькие и размещаются в несколько LUT), при двойном клике раскрываются ее свойства, в свойствах есть возможность просмотреть: логическую схему, логическое выражение, таблицу истинности, карту Карно (рис. 1.18).

Просмотр RTL схем используется для визуального выявления ошибок проектирования на раннем этапе. Как правило, такой подход выявляет наличие\отсутствие какого либо модуля, наличие\отсутствие или правильность соединений. Для более глубокого анализа схем используется программное моделирование.

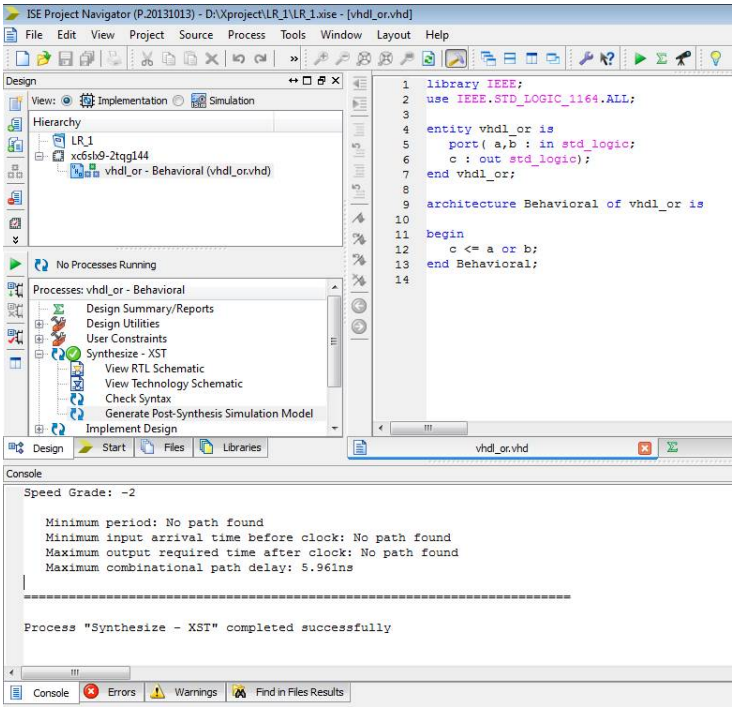


Рис. 1.14. Окно ProjectNavigator САПР XilinxISE с выводом положительного результата синтеза

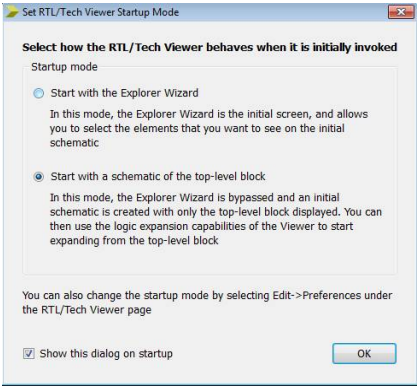


Рис. 1.15. Окно в ProjectNavigator САПР XilinxISE выбора вида вывода RTL схемы

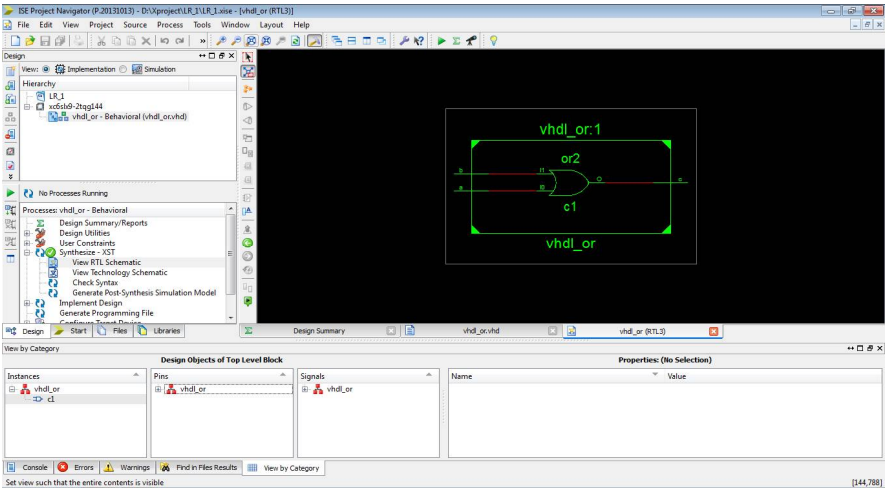


Рис. 1.16. Окно ProjectNavigator САПР XilinxISE
с выводом RTL схемы

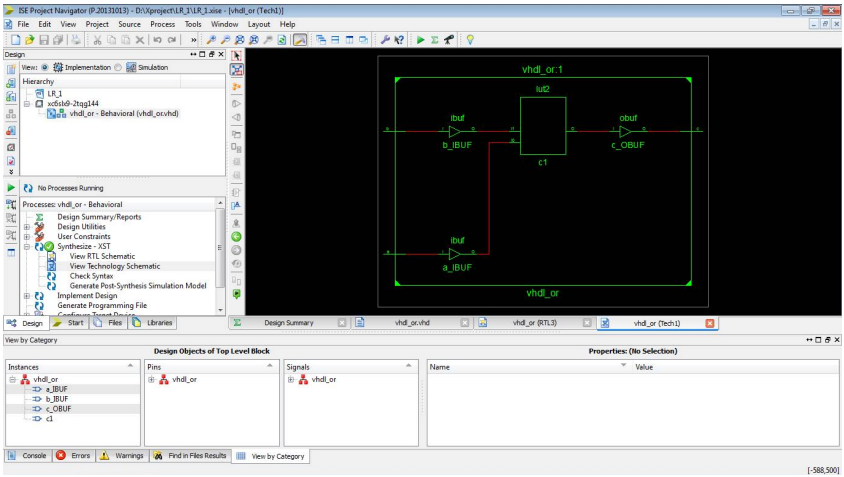
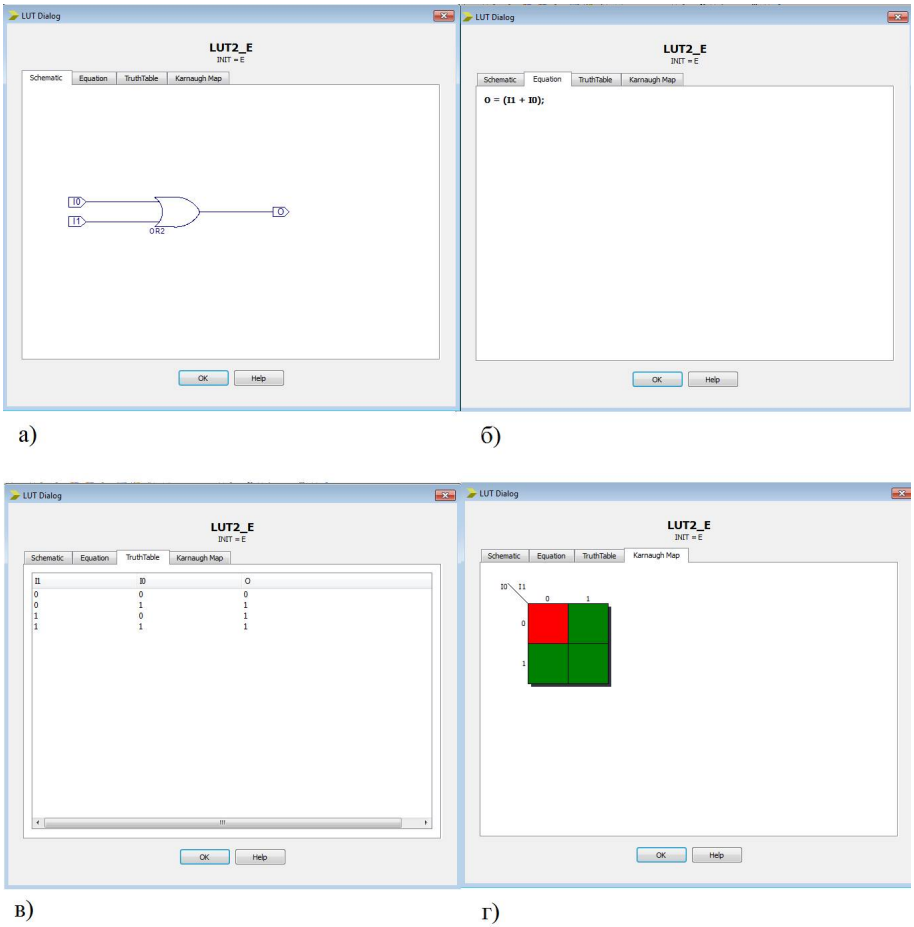


Рис. 1.17. Окно ProjectNavigator САПР XilinxISE с выводом RTL
схемы на стандартных элементах ПЛИС



а) б)

в) г)

Рис. 1.18. Окно свойств LUT ячейки: а) логическая схема; б) логическое выражение; в) таблица истинности; г) карта Карно

1.4. Работа в схемотехническом редакторе САПР Xilinx ISE

САПР Xilinx ISE DS содержит встроенный схемотехнический редактор. Данный редактор позволяет создавать цифровые схемы разной сложности и уровня. Схемотехнический редактор работает со специальными файлами формата «SCH» представляющие собой текстовый файл подобный HTML страницам. Взаимодействие с синтезатором выполняется посредством пере-

вода «SCH» файла в HDL код. Таким образом, схемотехнический редактор является надстройкой над HDL, позволяющей более удобно, в визуальном плане, проектировать цифровые схемы малого и среднего уровня. При разработке больших схем, возможности схемотехнического редактора нивелируются перед разработкой только на HDL.

Создадим новый пустой проект в папке D:\Xproject. Добавим новый файл формата «SCH», выбираем Project/New Source. В типе файлов выбираем «Schematic», задаем имя. Затем проверяем суммарные настройки, подтверждаем создание файла, нажав «Finish». Сразу установим полученный файл как файл верхнего уровня иерархии Source/Set as top module.

С добавлением нового схемотехнического файла добавляются закладки символов (Symbols) (рис. 1.19) и настроек (Options). Во вкладке настроек изменяются параметры работы схемотехнического редактора. Во вкладке символов отображаются все доступные символы проекта. Вкладка разделена на два окна: окно категорий, разделяющее все символы по функционалу и окна отображения символов в конкретной категории. Имеется строка поиска, упрощающая поиск конкретного символа. На основе модулей написанных на HDL или IP ядер можно создавать символы для схемотехнического редактора.

Символ это графического отображение, используемое при построении изображения схемы, практически все символы несут под собой модуль HDL описания используемый на этапе синтеза (в САПР Xilinx ISE есть символы доступные только в схемотехническом редакторе, возможности обратиться к описанию из HDL языка для таких символов нет, такими являются специальные модули ввода-вывода и базовые логические элементы).

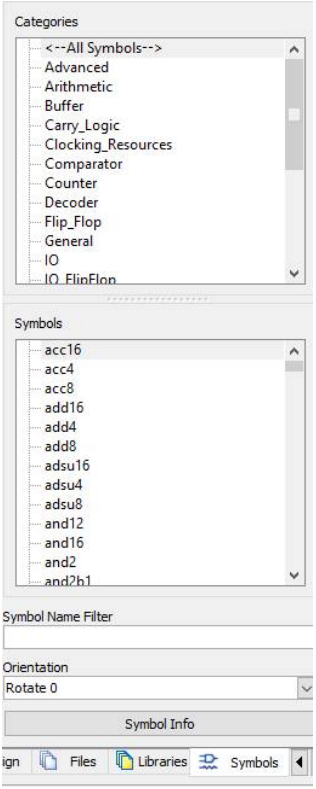


Рис. 1.19. Вкладка символов

Пусть необходимо построить логическую схему 2ИЛИ.

Находим необходимые символы, выбрав категорию или найдя через строку поиска (пользоваться строкой поиска намного удобнее, например, введя «and» в окно отображения символов выведутся все символы, в имени которых есть «and», учтите что поиск ведется по категории, которая выбрана в окне категорий, для глобального поиска выбирайте «All Symbols»). Для построения схемы находим элементы «or2».

Двойной клик зафиксирует элемент как выбранный, при наведении на главное поле в файле схемотехнического редактора с зафиксированным элементом он будет следовать за курсором. Одиночный клик в поле файла установит элемент там, где

он отображен, можно создавать неограниченное количество элементов в поле файла, элементы не могут наслаиваться друг на друга.

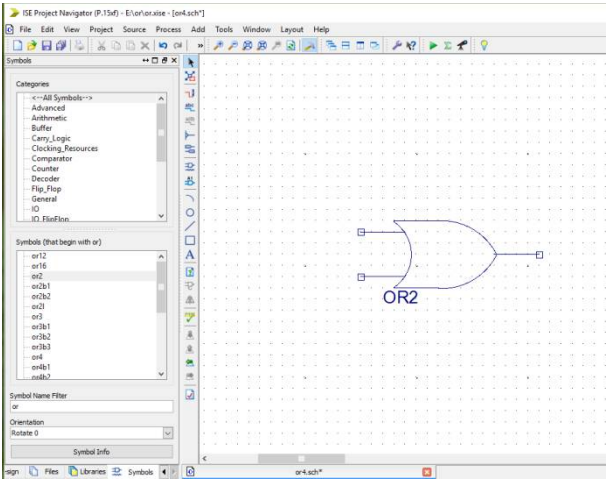


Рис. 1.20. Окно редактора схемотехнических файлов с добавленным символом

Добавляем буфер Buffer, Buf. Создаем межсоединения (выбираем add/wire,или нажав CTRL+W), или кликнув на пиктограмму  в панели быстрых действий). Курсор в поле изменит вид на перекрестие. Для соединения наведем перекрестие на контакт одного из символов, при этом контакт обрисуется по углам четырьмя квадратами, указывая на то, что межсоединение начнется с контакта. Зажимаем левую кнопку мыши и протягиваем курсор до необходимого контакта.

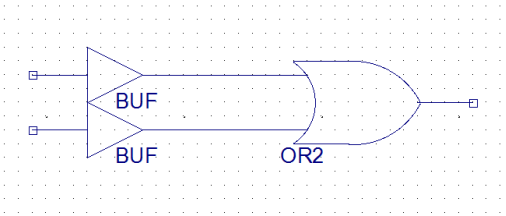


Рис. 1.21. Установленное межсоединение

Добавляем входы и выходы схемы, выбираем маркер add I/O marker. Наводим маркер на конец межсоединения и устанавливаем его левой кнопкой мыши. При установке входа\выхода имя присваивается автоматически, для наглядности переименуем с соответствующим логическим выражением. Выбираем порт и в контекстном меню выбираем «Rename port». Изменяем значения для всех портов.

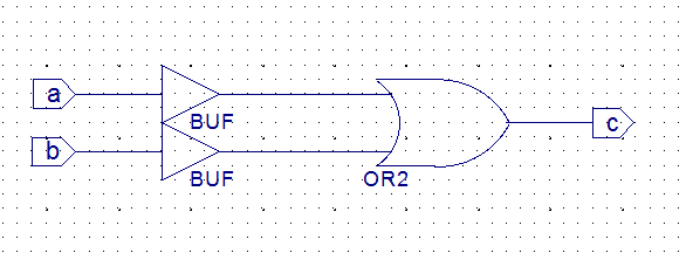


Рис. 1.22. Схема с переименованными портами

1.5. Программное моделирование работы схемы

Программное моделирование используется для выявления ошибок в проекте. Программное моделирование в современных САПР позволяет провести симуляцию схемы на различных уровнях, начиная с поведенческого моделирования (для проверки только алгоритма работы), до симуляции процессов протекающих в микросхеме ПЛИС с учетом задержек между элементами и т.д. Для конкретно данной лабораторной работы

нас интересует только поведенческое тестирование. Необходимо перейти в режим симуляции для этого выбираем во вкладке Design в самом верху возле «View» элемент Simulation (в противовес Implementation). В меню выбора типа симуляции выбираем Behavivoral (поведенческое) (рис. 1.23).

Создаем файл тестбенча, выбираем Project\NewSource, как и при создании обычного файла: тип модуля – «VHDL testbench» задаем имя. В следующем окне выбираем модуль для тестирования. Как правило, САПР не только просто создает файл и добавляет в него выбранный элемент как модуль, но и создает дополнительные конструкции, предназначенные для тестирования, как правило, это конструкции для входных значений и тактовый генератор.

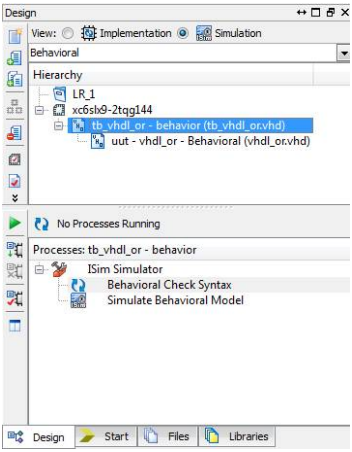


Рис. 1.23. Вкладка Design в режиме симуляции

Листинг созданного тестбенча.

```
LIBRARYieee;
USE ieee.std_logic_1164.ALL;
ENTITY tb_vhdl_or IS
END tb_vhdl_or;
ARCHITECTURE behavior OF tb_vhdl_or IS
   -- Component Declaration for the Unit Under Test (UUT)
```

```
   COMPONENTvhdl_or                    --описание модуля
входящего в тестбенч
PORT(
a : IN  std_logic;
b : IN  std_logic;
c : OUT  std_logic
);
ENDCOMPONENT;                          -- конец описания мо-
дуля входящего в тестбенч
--Inputs                               -- сигналы входов
signal a : std_logic := '0';
signal b : std_logic := '0';
        --Outputs                      -- сигналы выходов
signal c : std_logic;
        constant<clock>_period : time := 10 ns;      --константа пе-
                                       риода генератора
BEGIN
        -- Instantiate the Unit Under Test (UUT)
uut: vhdl_or PORTMAP (                 --добавление модуля
входящего в тестбенч
a => a,
b => b,
c => c
      );
-- Clockprocessdefinitions             --описание тактового генера-
тора
<clock>_process :process
begin
                <clock><= '0';
                wait for <clock>_period/2;
                <clock><= '1';
                wait for <clock>_period/2;
end process;
-- Stimulusprocess
stim_proc: process                     --описание процесса для
                                       входных значений
begin
```

```
    -- hold reset state for 100 ns.
wait for 100 ns;
wait for <clock>_period*10;
    -- insert stimulus here
wait;
end process;
END;
```

Так как нет необходимости в использовании тактового генератора, стираем код с элементами его описания: константа периода и процесс. В описание процесса входных значений добавляем перебор всех вариантов как в таблице истинности. Между описаниями состояний добавим элементы задержки для выдержки сигнала в процессе симуляции. После корректировки код примет вид представленный ниже. В процессе перебираются все вариации входных значений для элемента «ИЛИ».

Сохраняем и проверяем тестбенч на синтаксические ошибки. После проверки или исправления запускаем процесс поведенческой симуляции, выбираем в меню процессов процесс поведенческой симуляции. Если до этого не был проведен процесс синтеза файла верхнего уровня иерархии то он автоматически запустится, затем пройдет программный расчет схемы.

Результаты выводятся в отдельном окне программы ISim, (рис. 1.24). Как видно из результатов схема работает корректно, вывод соответствует работе элемента «ИЛИ».

Листинг кода тестбенча после корректировки.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY tb_vhdl_or IS
END tb_vhdl_or;
ARCHITECTURE behavior OF tb_vhdl_or IS
   COMPONENT vhdl_or
PORT(
a : IN  std_logic;
b : IN  std_logic;
c : OUT  std_logic
       );
   END COMPONENT;
signal a : std_logic := '0';
signal b : std_logic := '0';
signal c : std_logic;
BEGIN
uut: vhdl_or PORT MAP (
a => a,
b => b,
c => c
       );
stim_proc: process
begin
a<= '0';              -- установка значений на каждый вход
b <= '0';
wait for 100 ns; -- задержка значений на 100 наносекунд.
a <= '1';
b <= '0';
wait for 100 ns;
a <= '0';
b <= '1';
wait for 100 ns;
a <= '1';
b<= '1';
wait;                                   --остановка процесса
          --при отсутствии процесс будет проводится циклично по кругу
endprocess; END;
```

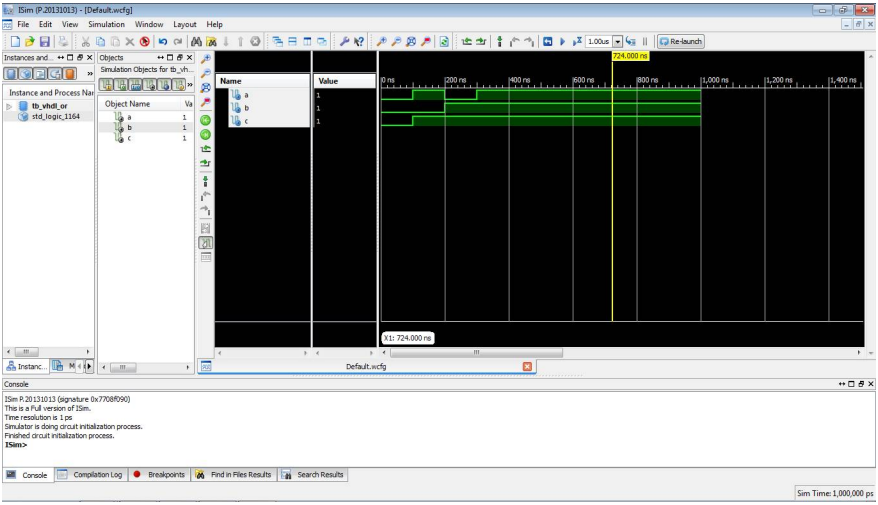


Рис. 1.24. Окно ISim с результатами симуляции

2. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Лабораторная работа № 1

СИНТЕЗ ЛОГИЧЕСКИХ СХЕМ ПО ЗАДАННОЙ ТАБЛИЦЕ ИСТИННОСТИ НА ОСНОВЕ ПЛИС

Цель работы

1. Получение практических навыков в разработке и исследовании логических элементов на основе ПЛИС.

2. Получение практических навыков в разработке и исследовании логических схем на основе ПЛИС.

3. Приобретение практических навыков использования системы виртуального схемотехнического моделирования Xilinx ISE Design Suite 14.1.

Содержание работы

1. Исследование логических элементов.

2. Разработка логической схемы по таблице истинности в заданном базисе.

Литература

1. Бибило П.Н. Основы языка VHDL: Учебное пособие. Изд. 6-е. - М.: Книжный дом «ЛИБРОКОМ», 2014. – 328 с.

2. Бабак В. П., Корченко А. Г., Тимошенко Н. П., Филоненко С. Ф. VHDL. Справочное пособие по основам языка – М.: Издательский дом «Додэка-XXI», 2008. – 224 с.

3. Тарасов И.Е. Разработка цифровых устройств на основе ПЛИС Xilinx с применением языка VHDL. –Изд. 2-е. – М.: Горячая линия – Телеком, 2015. – 252 с.

Содержание отчета

Цель работы.

1. Исследование логических элементов (ЛЭ);

1.1 Таблица истинности ЛЭ в соответствии с индивидуальным заданием;

1.2 Реализация ЛЭ в схемотехническом редакторе Xilinx ISE Design Suite 14.1;

1.3 Проверка работоспособности ЛЭ в симуляторе ISim с приведением временных диаграмм;

1.4 Описание заданного логического элемента на языке VHDL;

1.5 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

1.6 Трансляция разработанного проекта, программирование ПЛИС;

2. Разработка логической схемы по заданной таблице истинности;

2.1 Таблица истинности в соответствии с индивидуальным заданием;

2.2 Переключательная функция (ПФ) в совершенной дизъюнктивной нормальной форме (СДНФ) и совершенной конъюнктивной нормальной форме (СКНФ). Минимизированная ДНФ (МДНФ) в заданном базисе;

2.3 Реализация МДНФ в схемотехническом редакторе Xilinx ISE Design Suite 14.1;

2.4 Проверка работоспособности в симуляторе ISim с приведением временных диаграмм;

2.5 Описание ПФ в МДНФ на языке VHDL;

2.6 Проверка работоспособности в симуляторе ISim с приведением временных диаграмм;

2.7 Трансляция разработанного проекта, программирование ПЛИС.

3. Выводы.

Краткие теоретические сведения

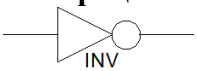
Для работы с базовыми логическими элементами в Xilinx ISE Design Suite 14.1. используется группа Logic, к ним относятся элементы И, ИЛИ, НЕ (инвертор), ИСКЛЮЧАЮЩЕЕ ИЛИ.

Таблица 2.1

Логические элементы группы Logic

Логическая операция НЕ

Y= not A



A	$Y = \overline{A}$
0	1
1	0

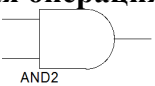
Логическая операция ИСКЛЮЧАЮЩЕЕ ИЛИ

Y= A xor B

A	B	Y=A^B
0	0	0
0	1	1
1	0	1
1	1	0

Логическая операция И

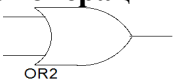
Y=A and B



A	B	Y=A·B
0	0	0
0	1	0
1	0	0
1	1	1

Логическая операция ИЛИ

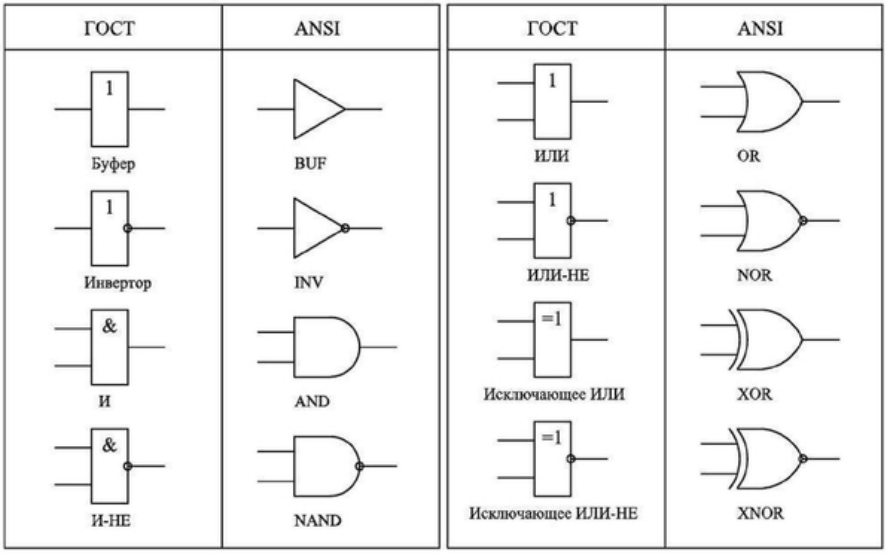
Y=A or B



A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

Таблица 2.2

Таблица соответствия обозначения ЛЭ



При преобразованиях логических выражений используются законы алгебры логики (табл. 2.3).

Таблица 2.3

Законы алгебры логики

Название закона	**для И**	**для ИЛИ**
Двойное отрицание	$\overline{\overline{A}}=A$	
Дополнительности	$A\cdot\overline{A}=0$	$A+\overline{A}=1$
Первого множества	$A\cdot 0=0$	$A+0=A$
Универсального множества	$A\cdot 1=A$	$A+1=1$
Повторения	$A\cdot A=A$	$A+A=A$
Поглощения	$A\cdot(A+B)=A$	$A+A\cdot B=A$
Переместительный	$A\cdot B=B\cdot A$	$A+B=B+A$
Сочетательный	$A\cdot(B\cdot C)=(A\cdot B)\cdot C$	$A+(B+C)=(A+B)+C$
Распределительный	$A+B\cdot C=(A+B)\cdot(A+C)$	$A\cdot(B+C)=A\cdot B+A\cdot C$
Склеивания	$(A+B)\cdot(A+\overline{B})=A$	$A\cdot B+A\cdot\overline{B}=A$
де Моргана	$\overline{A\cdot B}=\overline{A}+\overline{B}$	$\overline{A+B}=\overline{A}\cdot\overline{B}$

Любая логическая схема описывается таблицей истинности. Для разработки требуемой логической схемы необходимо получить ПФ в СДНФ или в СКНФ.

Затем с целью упрощения цифрового устройства минимизируют его логическое выражение, по которому разрабатывают схему.

Функция в СДНФ представляет собой сумму элементарных произведений (минтермов). Приведем пример нахождения СДНФ для заданной таблица истинности (табл. 2.4).

Таблица 2.4

X1	X0	F
0	0	**1**
0	1	0
1	0	**1**
1	1	0

Для каждого набора переменных, при котором F равна 1, записывается произведение, в котором с отрицанием берутся переменные, имеющие значение 0.

$$F = \overline{X1} \cdot \overline{X0} + X1 \cdot \overline{X0}$$

Функция в СКНФ представляет собой произведение элементарных сумм (макситермов). Приведем пример нахождения СКНФ для заданной таблица истинности (табл. 2.5).

Таблица 2.5

X1	X0	F
0	0	1
0	1	**0**
1	0	1
1	1	**0**

Для каждого набора переменных, при котором F равна 0, записывается сумма, в котором с отрицанием берутся переменные, имеющие значение 1.

$$F = (\overline{X1} + X0) \cdot (\overline{X1} + \overline{X0})$$

Порядок выполнения работы

1. Исследование логических элементов
1.1. Таблица истинности ЛЭ в соответствии с индивидуальным заданием

Рассмотрим пример построения простейшего логического элемента 2И. Приведем таблицу истинности (табл. 2.6).

Таблица 2.6

Таблица истинности логического элемента 2И

X1	X2	Y1=X1·X2
0	0	0
0	1	0
1	0	0
1	1	1

1.2. Реализация в схемотехническом редакторе Xilinx ISE Design Suite 14.1

Схема логического элемента 2И в программной оболочке ISE представлена на рис. 2.1

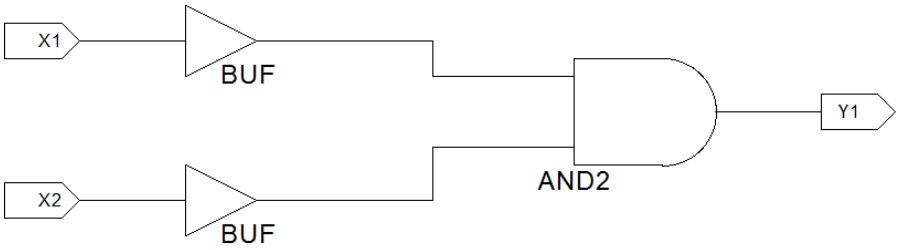


Рис. 2.1. Схема элемента 2И

1.3. Проверка работоспособности ЛЭ в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
 …
PROCESS
  BEGIN
        X1<='0'; // присваиваем переменной X1 значение 0
        X2<='0'; // присваиваем переменной X2 значение 0
        wait for 50 ns; // длительность 50 нс
  …                    //вписываем необходимые значения
    WAIT;
  END PROCESS;
```

1.4. Описание заданного логического элемента на языке VHDL

library IEEE; use IEEE.STD_LOGIC_1164.ALL;	Подключаем все объявления пакета STD_LOGIC_1164, входящего в библиотеку IEEE
entity VHDL_I is Port (X1 : in STD_LOGIC; X2 : in STD_LOGIC; Y1 : out STD_LOGIC); end VHDL_I;	Определяем сигналы, которыми объект будет обмениваться X1, X2 – входные порты, Y1 – выходной порт, std_logic - это просто провод.
architecture Behavioral of VHDL_I is begin Y1<=X1 and X2; end Behavioral;	Задаем алгоритм работы В выходной порт Y1 записывается значение X1·X2

1.5. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
…
process
  begin
X1<='0';
X2<='0';
 wait for 100 ns;
 …
     wait;
  end process;
```

2. Разработка логической схемы по заданной таблице истинности

2.1. Таблица истинности

Рассмотрим пример на основе заданной таблицы истинности.

Таблица 2.7

Таблица истинности

X1	X2	X3	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

2.2 ПФ в СДНФ и СКНФ. Минимизированная ДНФ (МДНФ) в заданном базисе

Представим ПФ в СДНФ

$Y = \overline{X1} \cdot \overline{X2} \cdot X3 + \overline{X1} \cdot X2 \cdot X3 + X1 \cdot \overline{X2} \cdot \overline{X3} + X1 \cdot X2 \cdot X3.$

Представим ПФ в СКНФ

$Y = (X1 + X2 + X3) \cdot (X1 + \overline{X2} + X3) \cdot (\overline{X1} + X2 + \overline{X3}) \cdot (\overline{X1} + \overline{X2} + X3).$

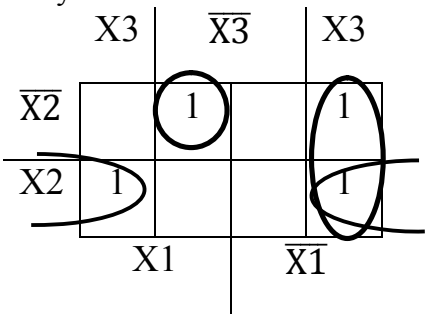
Применив закон склеивания для ПФ в СДНФ, получим

$Y = \overline{X1} \cdot \overline{X2} \cdot X3 + \overline{X1} \cdot X2 \cdot X3 + X1 \cdot \overline{X2} \cdot \overline{X3} + X1 \cdot X2 \cdot X3 =$

$= \overline{X1} \cdot X3 \cdot (\overline{X2} + X2) + X1 \cdot \overline{X2} \cdot \overline{X3} + X2 \cdot X3(\overline{X1} + X1) =$

$= \overline{X1} \cdot X3 + X1 \cdot \overline{X2} \cdot \overline{X3} + X2 \cdot X3$

Применив графический способ минимизации с помощью диаграммы Вейча, получим $Y = \overline{X1} \cdot X3 + X1 \cdot \overline{X2} \cdot \overline{X3} + X2 \cdot X3$.



На диаграмме изображены компактные группы $\overline{X1} \cdot X3 + X1 \cdot \overline{X2} \cdot \overline{X3} + X2 \cdot X3$.

Приведем к базису 2И-НЕ $Y = X1 \cdot X3 + X1 \cdot X2 \cdot X3 + X2 \cdot X3 =$ $= \overline{\overline{X1}\,\overline{X3}} \cdot \overline{X1 \cdot \overline{X2} \cdot \overline{X3}} \cdot \overline{X2 \cdot X3} =$ $\overline{X1}X3 \cdot X1\overline{X2}\overline{X3} \cdot X2X3$	Приведем к базису 2ИЛИ-НЕ $Y = \overline{\overline{X1 \cdot X3} + \overline{X1 \cdot \overline{X2} \cdot \overline{X3}} + \overline{X2 \cdot X3}} =$ $\overline{X1}+X3+\overline{X1}+X2+X3+\overline{X2}+\overline{X3}$

2.3. Реализация ПФ в схемотехническом редакторе Xilinx ISE Design Suite 14.1

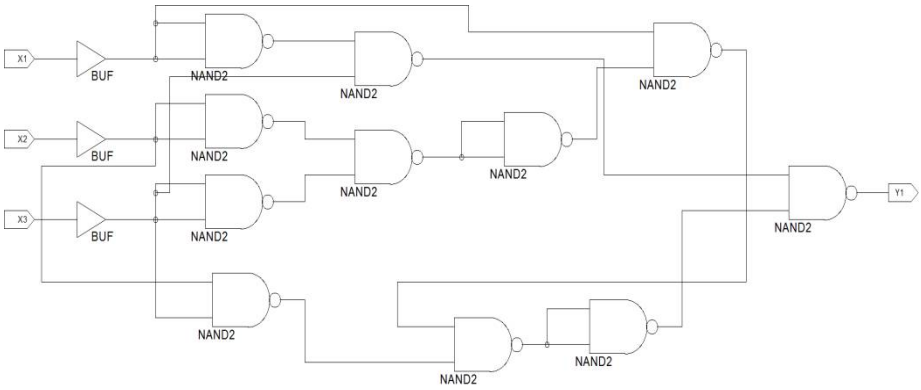


Рис. 2.2. Схема ПФ в базисе 2И-НЕ

2.4. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
…
process
  begin
X1<='0';
X2<='0';
X3<='0';
 wait for 100 ns;
 …
     wait;
  end process;
```

2.5. Описание ПФ в МДНФ на языке VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity PF_VHDL is
    Port ( X1 : in  STD_LOGIC;
           X2 : in  STD_LOGIC;
           X3 : in  STD_LOGIC;
           Y1 : out  STD_LOGIC);
end PF_VHDL;

architecture Behavioral of PF_VHDL is

begin
Y1<=(not(X1)and X3) or (X1 and not(X2) and not(X3)) or (X2 and X3);

end Behavioral;
```

2.6. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
…
process
   begin
X1<='0';
X2<='0';
X3<='0';
 wait for 100 ns;
 …
      wait;
   end process;
```

Индивидуальные задания

№ варианта	Заданный ЛЭ	Заданная таблица истинности				Базис
1	3И	x1	x2	x3	y	2И-НЕ
		0	0	0	0	
		0	0	1	0	
		0	1	0	1	
		0	1	1	1	
		1	0	0	1	
		1	0	1	0	
		1	1	0	0	
		1	1	1	1	
2	2И-НЕ	x1	x2	x3	y	3И-НЕ
		0	0	0	0	
		0	0	1	0	
		0	1	0	0	
		0	1	1	1	
		1	0	0	1	
		1	0	1	1	
		1	1	0	0	
		1	1	1	1	
3	3И-НЕ	x1	x2	x3	y	2И-НЕ
		0	0	0	1	
		0	0	1	0	
		0	1	0	1	
		0	1	1	0	
		1	0	0	0	
		1	0	1	0	
		1	1	0	1	
		1	1	1	1	

4	2ИЛИ-НЕ	x1	x2	x3	y	3И-НЕ
		0	0	0	0	
		0	0	1	0	
		0	1	0	1	
		0	1	1	1	
		1	0	0	1	
		1	0	1	1	
		1	1	0	0	
		1	1	1	0	
5	2ИЛИ	x1	x2	x3	y	2ИЛИ-НЕ
		0	0	0	0	
		0	0	1	1	
		0	1	0	0	
		0	1	1	0	
		1	0	0	1	
		1	0	1	0	
		1	1	0	1	
		1	1	1	1	
6	3ИЛИ	x1	x2	x3	y	2И-НЕ
		0	0	0	0	
		0	0	1	0	
		0	1	0	1	
		0	1	1	0	
		1	0	0	1	
		1	0	1	0	
		1	1	0	1	
		1	1	1	1	

7	2 ИСКЛ ИЛИ	x1	x2	x3	y	3И-НЕ
		0	0	0	1	
		0	0	1	0	
		0	1	0	0	
		0	1	1	1	
		1	0	0	0	
		1	0	1	0	
		1	1	0	1	
		1	1	1	1	
8	3ИСКЛ ИЛИ	x1	x2	x3	y	2И-НЕ
		0	0	0	0	
		0	0	1	1	
		0	1	0	1	
		0	1	1	1	
		1	0	0	0	
		1	0	1	1	
		1	1	0	0	
		1	1	1	0	
9	4ИЛИ-НЕ	x1	x2	x3	y	3ИЛИ-НЕ
		0	0	0	0	
		0	0	1	1	
		0	1	0	0	
		0	1	1	1	
		1	0	0	0	
		1	0	1	1	
		1	1	0	0	
		1	1	1	1	
10	4ИЛИ	x1	x2	x3	y	2ИЛИ-НЕ
		0	0	0	0	
		0	0	1	1	
		0	1	0	0	
		0	1	1	0	
		1	0	0	1	
		1	0	1	0	
		1	1	0	1	
		1	1	1	1	

Контрольные вопросы

1. Правила перевода из десятичной системы счисления (СС) в двоичную СС и наоборот.
2. Правила перевода из шестнадцатеричной системы счисления (СС) в двоичную СС и наоборот.
3. Правила перевода из восьмеричной системы счисления (СС) в двоичную СС и наоборот.
4. Основные операции и законы алгебры логики. Порядок выполнения операций в сложных функциях.
5. Основные теоремы и алгебры логики для функций одной и двух переменных.
6. Таблицы истинности ЛЭ ИЛИ, И, НЕ.
7. Способы задания переключательной функции: словесный, табличный, алгебраический.
8. Две основные алгебраические формы ПФ: дизъюнктивная и конъюнктивная.
9. Реализация схемы ПФ в любом базисе, универсальном базисе 2И-НЕ, 2ИЛИ-НЕ.
10. Минимизация ПФ методом алгебраических преобразований.
11. Минимизация ПФ с помощью карт Карно.
12. Минимизация ПФ с помощью диаграмм Вейча.

Лабораторная работа № 2

РАЗРАБОТКА ТРИГГЕРОВ НА ОСНОВЕ ПЛИС

Цель работы

1. Получение практических навыков в разработке и исследовании триггеров на основе ПЛИС.

2. Приобретение практических навыков использования системы виртуального схемотехнического моделирования Xilinx ISE Design Suite 14.1.

Содержание работы

1. Исследование D триггера.

2. Исследование JK триггера.

Литература

1. Бибило П.Н. Основы языка VHDL: Учебное пособие. Изд. 6-е. - М.: Книжный дом «ЛИБРОКОМ», 2014. – 328 с.

2. Бабак В. П., Корченко А. Г., Тимошенко Н. П., Филоненко С. Ф. VHDL. Справочное пособие по основам языка – М.: Издательский дом «Додэка-XXI», 2008. – 224 с.

3. Тарасов И.Е. Разработка цифровых устройств на основе ПЛИС Xilinx с применением языка VHDL. –Изд. 2-е. – М.: Горячая линия – Телеком, 2015. – 252 с.

Содержание отчета

Цель работы.

1. Исследование D - триггера;

1.1 Таблица истинности D триггера;

1.2 Реализация D триггера в схемотехническом редакторе Xilinx ISE Design Suite 14.1;

1.3 Проверка работоспособности в симуляторе ISim с приведением временных диаграмм;

1.4 Описание D триггера на языке VHDL;

1.5 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

1.6 Трансляция разработанного проекта, программирование ПЛИС;

2. Исследование JK - триггера;

2.1 Таблица истинности JK триггера;

2.2 Реализация JK триггера в схемотехническом редакторе Xilinx ISE Design Suite 14.1;

2.3 Проверка работоспособности в симуляторе ISim с приведением временных диаграмм;

2.4 Описание JK триггера на языке VHDL;

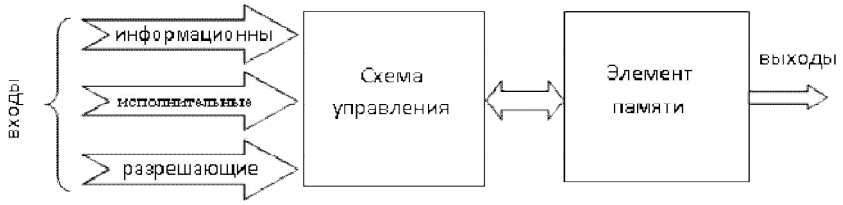
2.5 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

2.6 Трансляция разработанного проекта, программирование ПЛИС;

3. Выводы.

Краткие теоретические сведения

Под памятью триггера понимается его способность учитывать свое предшествующее состояние.



Наличие памяти обусловливает необходимость анализа состояния устройства в двух соседних тактах работы. Для сигналов, действующих в этих тактах, принято к наименованию сигнала (вывода) добавлять соответствующий индекс такта. Так, запись $Q^t=1$ означает, что на выходе Q до момента времени t действует логическая единица, а запись $Q^{t+1}=0$ означает, что с момента времени t на выходе Q действует логический ноль. Понятно, что изменение состояния схемы происходит в момент времени t. Рассматривая Q как выход триггера можно выделить основные режимы работы триггеров.

1. $Q^{t+1}=Q^t$ - режим хранения информации.
2. $Q^{t+1}=\overline{Q}^t$ - режим переключения триггера в противоположные состояния.
3. $Q^{t+1}=1$ - установка (хранение) триггера в «1».
4. $Q^{t+1}=0$ - установка (хранение) триггера в «0».
5. $Q^{t+1}=X$ - состояние выхода не определено, для триггеров этот режим запрещенный.

В общем случае триггер содержит элемент памяти и схему управления этим элементом (или несколькими элементами).

Информационные входы определяют, какая информация будет записана в триггер:

а) S, J - входы установки $Q^{t+1}=1$;

б) R, K - входы установки $Q^{t+1}=0$;

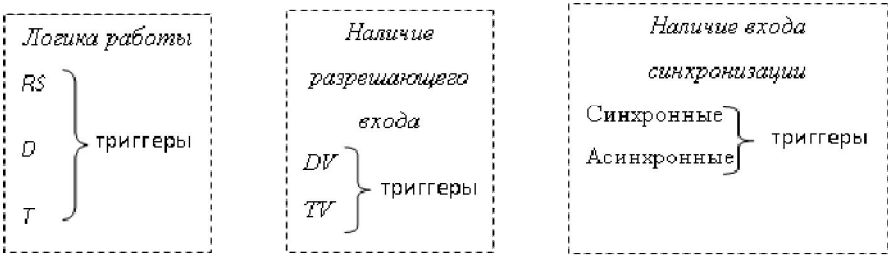
в) T - счетный вход: $Q^{t+1}=$, состояние триггера меняется на противоположное;

г) D-вход: $Q^{t+1}=D^t$, в триггере запоминается значение D^t.

Разрешающий V-вход разрешает прием сигналов по информационным входам.

Исполнительный С - вход (синхронизации) разрешает формирование выходных Q-сигналов.

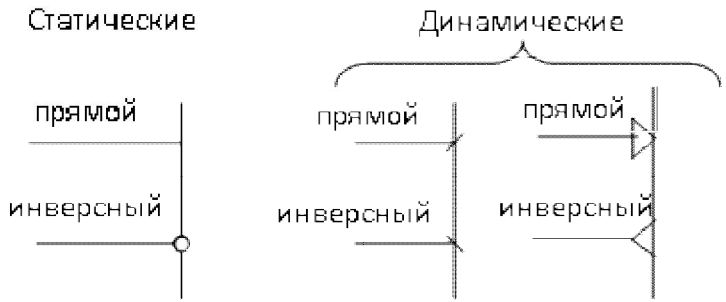
Особо выделяется триггер Шмитта, имеющий также два устойчивых состояния, как и все другие триггеры, но не обладающий памятью. Чаще всего применяется в качестве пороговых устройств, так как имеет гистерезисную передаточную характеристику и поэтому позволяет повысить помехоустойчивость этих устройств.



В основу классификации триггеров положена логика работы; наличие входов разрешения записи V , считывания С, предварительной установки.

Входы (выходы) могут быть прямые и инверсные, статические и динамические.

Обозначение входов



В статических входах информация воспринимается соответствующим уровнем сигнала: во время действия логической единицы прямой статический вход активизируется и отрабатывается его логика работы; инверсный статический вход активизируется уровнем логического нуля. Следует отметить, что статические входы активны все время, пока действует соответствующий уровень напряжения. В динамических входах управляющими являются перепады напряжения: перепад с нуля в единицу (фронт импульса) активизирует прямой динамический вход, и наоборот. RS-триггер имеет два информационных входа S (set) – установка $Q^{t+1}=1$, R (reset) - сброс единицы, т.е. $Q^{t+1}=0$. Таким образом, $Q^{t+1}=F(S^t, R^t, Q^t)$.

Таблица 2.8

Таблица истинности RS триггера

№ набора	S^t	R^t	Q^t	Q^{t+1}	режим
0	0	0	0	0	Хранение
1	0	0	1	1	Хранение
2	0	1	0	0	Хранение
3	0	1	1	0	Переключение
4	1	0	0	1	Переключение
5	1	0	1	1	Хранение
6	1	1	0	Н	Запрещен
7	1	1	1	Н	Запрещен

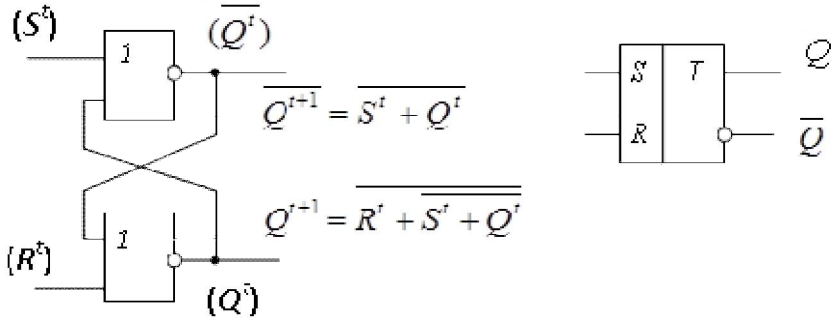


Рис. 2.3. RS-триггер на элементах ИЛИ-НЕ

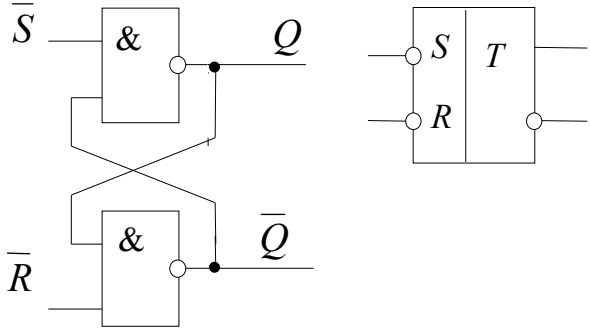


Рис. 2.4. RS-триггер на элементах И-НЕ и его условное обозначение

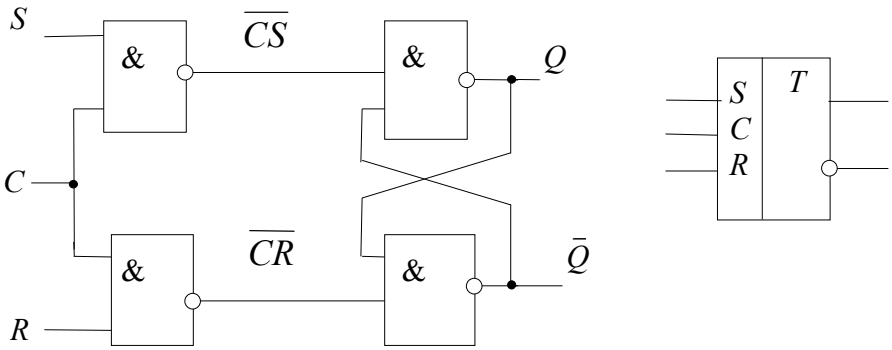


Рис. 2.5. Схема синхронного RS-триггера

При C=1 триггер работает как асинхронный с прямыми входами, при C=0, т.е. триггер хранит информацию независимо от состояния информационных входов R и S. Как и в предыдущем случае C=R=S=1 запрещен.

Согласно логике работы D-триггера, он может быть только синхронным, в противном случае теряется основное свойство триггера – память и триггер работает как повторитель D-входа. Для реализации свойств триггера необходимо управлять прохождением информационного сигнала в триггере, т.е. дополнить

триггер входом синхронизации *C*. При C=1 осуществляется запоминание информации с D- входа, а при C=0 - ее хранение.

Таблица 2.9

Таблица истинности D-триггера

№ наб	C^t	D^t	Q^t	Q^{t+1}	
0	0	0	0	0	$C^t=0$ Хранение $Q^{t+1}= Q^t$
1	0	0	1	1	
2	0	1	0	0	
3	0	1	1	1	
4	1	0	0	0	$C^t=1$ D-логика $Q^{t+1}= D^t$
5	1	0	1	0	
6	1	1	0	1	
7	1	1	1	1	

Для реализации D-триггера на основе RS- триггера необходимо на входы RS- триггера подать сигналы в противофазе, тогда D = и, следовательно, схема *D*-триггера имеет вид, представленный на рис. 2.6.

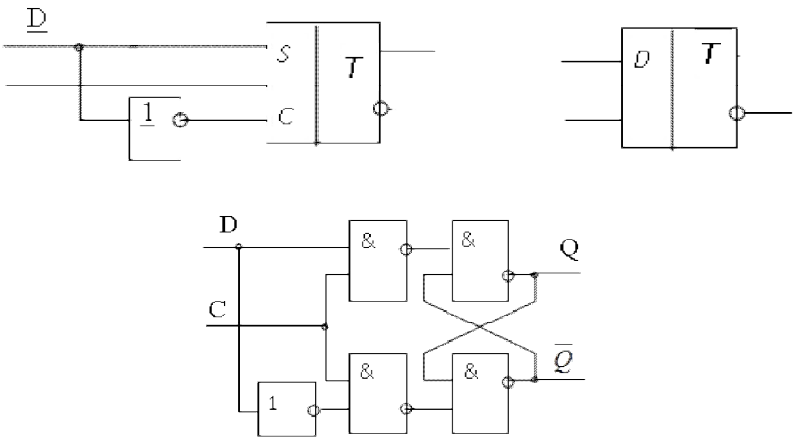


Рис. 2.6. Реализация D-триггера на основе RS-триггера и его условное обозначение

Второй вариант триггера с одним информационным входом может быть реализован, если входной сигнал x считать исполнительным, а информационным сигналом считать выходной сигнал самого триггера.

При поступлении входного сигнала x(T)=1 триггер в соответствии с логикой работы должен менять свое состояние на противоположное. Переключательная функция *T*-триггера $Q^{t+1} = T^t\overline{Q^t} + \overline{T^t}Q^t$.

При $T^t = 1, Q^{t+1} = \overline{Q^t}$, т.е. можно сказать, что в очередном такте (t+1) T-триггер должен запоминать свое предыдущее состояние (момент времени t) в инвертированном виде. Такая формулировка логики работы T-триггера позволяет достаточно просто получить его схему на основе схемы D-триггера.

Действительно, в этом случае при T=1, $Q^{t+1} = D^t = \overline{Q^t}$. Но, к сожалению, при длительности входного сигнала большей времени задержки переключения триггера $T>t_{з\ пер}$ изменение состояния инверсного выхода триггера передается на его D-вход, что с небольшой задержкой $t_{з.пер}$ приводит вновь к изменению состояния инверсного выхода триггера на противоположное и т.д. Таким образом, схема будет самопроизвольно переключаться с частотой $1/t_{з\ пер}$. Выход из этого противоречия возможен организацией задержки поступления выходного сигнала $\overline{Q}^t$ на D-вход на время $t_{зад}>T$. Тогда при поступлении T=1 триггер переключится в противоположное состояние, однако, до поступления T=0 информация об изменившемся выходном сигнале $\bar{Q}$ не поступает на вход D. Однако это очень неудобно с конструктивной точки зрения, и в ряде случаев просто нереализуемо (поскольку длительность импульса T может быть очень большой). Решение данной проблемы заключается в использовании для построения T-триггеров D-(RS)- триггеров с динамическими синхровходами. Один из вариантов получения триггеров с динамическими синхровходами заключается в использовании так называемой двухступенчатой структуры триггеров.

Таблица 2.10

Таблица истинности Т-триггера

№ наб	T^t	Q^t	Q^{t+1}
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

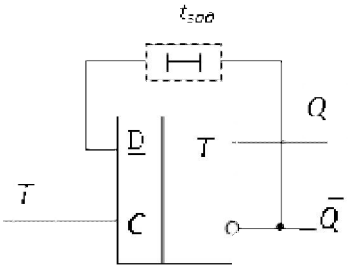


Рис. 2.7

В этом случае триггер строится из двух последовательно включенных триггеров со статическими синхровходами, причем на синхровход триггера первой ступени сигнал синхронизации подается непосредственно, а на синхровход триггера второй ступени сигнал синхронизации подается через инвертор. Аналогично строятся двухступенчатые RS-триггеры. При поступлении сигнала синхроимпульса C=1 входная информация записывается по D-входу в триггер первой ступени. Хотя на выходе триггера первой ступени во время действия синхроимпульса устанавливается состояние заданное по D-входу, в триггер второй ступени запись невозможна т.к. в это время $\bar{C}$=0. При смене на C-входе логической единицы логическим нулем информация с выхода триггера первой ступени переписывается в триггер второй ступени и поступает на его выход. Но с этого же момента триггер первой ступени уже не воспринимает входную информацию. Таким образом, на выходе двухступенчатого триггера устанав-

ливается информация, существующая на D-входе непосредственно перед поступлением среза синхроимпульса. Принято говорить, что такой триггер срабатывает по срезу синхроимпульса. Синхровход триггера обозначается как инверсный динамический и является счетным Т-входом. В целом такой триггер является асинхронным Т-триггером и имеет условное графическое обозначение.

Таблица 2.11

Таблица истинности синхронного Т – триггера

№ наб.	T^t	C^t		Q^t	Q^{t+1}	режим
0	0	0	X	0	0	Хранение $Q^{t+1}= Q^t$
1	0	0	X	1	1	
2	0	1	X	0	0	
3	0	1	X	1	1	
4	1	0	X	0	0	
5	1	0	X	1	1	
6	1	1		0	1	Переключение $Q^{t+1}=\overline{Q^t}$
7	1	1		1	0	

Для входа *С* приведены два столбца – один традиционный, другой (выделенный) часто используется в справочниках. При этом символ X означает безразличное состояние входа (0 или 1), а символ использованный на наборах 6 и 7 означает поступление среза синхроимпульса. Кроме того, состояние выходов также может быть отражено в аналитическом виде так, как показано в столбце «режим». Принципиальная схема синхронного Т-триггера на основе двухступенчатого RS-триггера приведена на рис. 2.7. Логика Т-триггера обеспечивается подачей сигналов обратной связи с выхода Q на вход R и с выхода $\bar{Q}$ на вход S. Фактически схема повторяет схему построения Т-триггера на основе одноступенчатых D-триггеров. Действительно, ведомый RS-триггер получает сигнал парафазно с выходов ведущего RS-

триггера, что соответствует логике работы одноступенчатого D-триггера. Ведущий RS-триггер получает сигнал парафазно с выходов ведомого RS-триггера, но прямой выход подключен к R-входу, а инверсный – к входу S. Это означает, что ведущий триггер используется также как D-триггер, но информацию запоминает в инвертированном виде. Т-вход разрешает прохождение сигналов в ведущем триггере, а С- вход разрешает оследовательное прохождение сигналов в ведущем триггере, а затем – в ведомом.

JK-триггер имеет входы установки (J) и сброса (К), подобные входам триггера RS. В отличие от последнего, допускает ситуацию с одновременной подачей сигналов на оба эти входа (J = К = 1). В этом режиме работает как счетный триггер относительно тактового входа. Триггер JK-типа имеет более сложную, по сравнению с RS-триггером, структуру и более широкие функциональные возможности. Помимо информационных входов J и К и прямого и инверсного выходов Q и $\bar{Q}$, JK-триггер имеет вход управления С (этот вход также называют тактирующим или счетным), а также асинхронные установочные R и S-входы. Обычно активными уровнями установочных сигналов являются нули. Установочные входы имеют приоритет над остальными. Активный уровень сигнала на входе S устанавливает триггер в состояние 0 = 1, а активный уровень сигнала на входе R - в состояние 0 = 0, независимо от сигналов на остальных входах. Если на входы установки одновременно подать пассивный уровень сигнала, то состояние триггера будет изменяться по фронту импульса на счетном входе в зависимости от состояния входов J и К.

Таблица 2.12

Таблица истинности JK-триггера

J	К	Q_t	Q_{t+1}
X	0	0	0
0	1	0	1
1	0	1	0
0	X	1	1

Работа JK-триггера описывается характеристическим уравнением $Q^{t+1} = \overline{Q^t} \cdot J + Q^t \overline{K}$.

Временные диаграммы приведены на рис. 2.6.

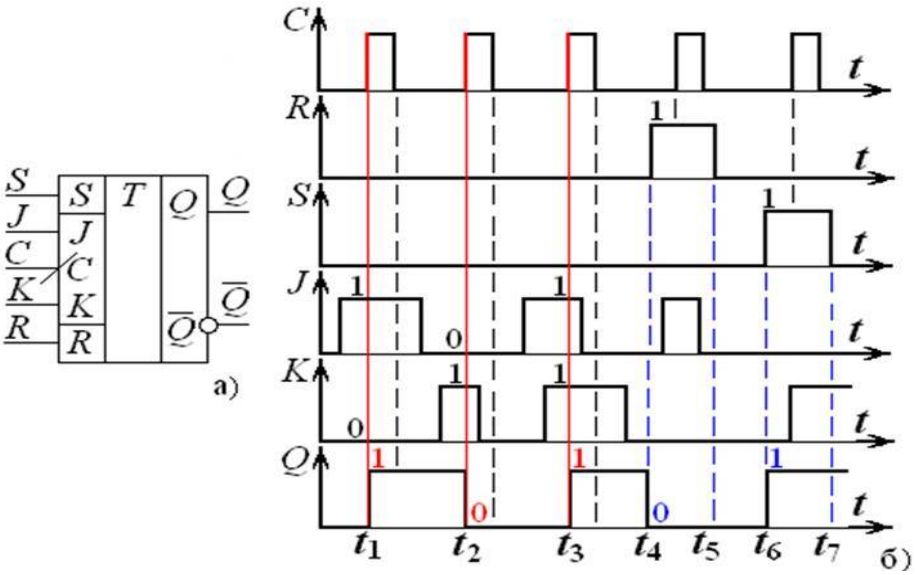


Рис. 2.8. Условно-графическое обозначение (а) и временная диаграмма (б) работы JK-триггера с асинхронными RS входами

Порядок выполнения работы

1. Исследование D - триггера
1.1. Таблица истинности D триггера

Clk	Data	Q_t
0	0	Q_{t-1}
0	1	Q_{t-1}
1	0	0
1	1	1

1.2. Реализация D триггера, тактируемого по переднему фронту в схемотехническом редакторе Xilinx ISE DS 14.1

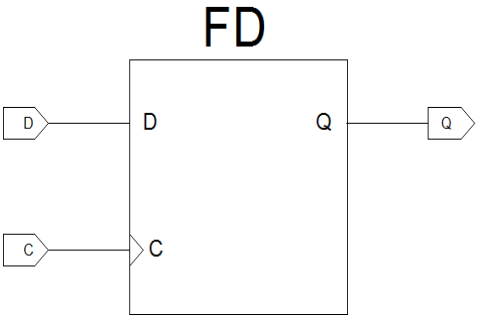


Рис. 2.9. Схема D триггера, тактируемого по переднему фронту, в схемотехническом редакторе Xilinx ISE Design Suite 14.1

1.3. Проверка работоспособности в симуляторе

Приведем пример определения входных сигналов в TestBench.

```
…
COMPONENT D_trigger
  PORT( D:IN STD_LOGIC;
        C:IN STD_LOGIC;
```

```
        Q:OUT STD_LOGIC);
  END COMPONENT;
  SIGNAL D:STD_LOGIC;
  SIGNAL C:STD_LOGIC;
  SIGNAL Q:STD_LOGIC;
  constant Clk_period : time := 100 ns;
BEGIN
  UUT: D_trigger PORT MAP(
D => D,
C => C,
Q => Q
  );
Clk_process :
process
  begin
C <= '0';
wait for Clk_period/2;
C<= '1';
wait for Clk_period/2;
end process;
      stim_proc: process
  begin
 D<='0';
 wait for 100 ns;
 D<='1';
 wait for 100 ns;
…
  wait;
  end process;
END;
```

1.4. Описание D триггера на языке VHDL

Для описания триггерных схем в VHDL используются операторы wait и if вместе с процессом, использующим аттрибуты переднего или заднего фронтов синхроимпульса.

Ниже приведены примеры создания описаний срабатывания по фронту:

(clk'event **and** clk='1') - аттрибут срабатывания по переднему фронту

(clk'event **and** clk='0') - аттрибут срабатывания по заднему фронту

rising_edge(clock) - вызов функции по переднему фронту

falling_edge(clock) - вызов функции по заднему фронту

На рис. 2.2 показаны процессы, происходящие в синхронных (тактируемых) триггерах.

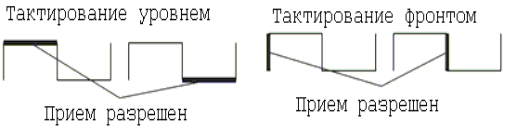


Рис. 2.10. Используемые обозначения синхросигналов на условно-графических обозначениях триггеров тактируемых уровнем или фронтом (срезом) синхросигнала

Записываем код для работы схемы для D триггера, тактируемого по передним фронту:

library IEEE; use IEEE.STD_LOGIC_1164.ALL;	Подключаем все объявления пакета std_logic_1164, входящего в библиотеку IEEE
entity D_trigger_VHDL is Port (D : in STD_LOGIC; C : in STD_LOGIC; Q : out STD_LOGIC); end D_trigger_VHDL;	Определяем сигналы, которыми объект будет обмениваться ***in*** – входные порты ***out*** – выходные порты
architecture Behavioral of D_trigger_VHDL is begin	Архитектурное тело Присвоение значения сигналу <=

process (C) begin
if (C'event and C = '1') then
Q<= D;
end if;
end process;
end Behavioral;
	В теле оператора ***process*** записываем последовательные операторы, при моделировании алгоритм в нем будет исполняться друг за другом после изменения одного из сигналов, перечисленных в списке инициализаторов

1.5. Проверка работоспособности в симуляторе ISim с приведением временной диаграммы

Приведем пример определения входных сигналов в TestBench.

```
…
--Inputs
  signal D : std_logic := '0';
  signal C : std_logic := '0';
 --Outputs
  signal Q : std_logic;

  constant C_period : time := 100 ns;
 BEGIN
 uut: D_trigger_VHDL PORT MAP (
        D => D,
        C => C,
        Q => Q
      );

  -- Clock process definitions
  C_process :process
  begin
C <= '0';
wait for C_period/2;
C <= '1';
wait for C_period/2;
```

```
  end process;
   -- Stimulus process
  stim_proc: process
  begin
D<='0';
 wait for 50 ns;
 D<='1';
 wait for 50 ns;
D<='0';
 wait for 20 ns;
 …
     wait;
  end process;
END;
```

1.6 Трансляция разработанного проекта, программирование ПЛИС

2. Исследование JK - триггера

2.1. Таблица истинности JK при C = 1

J	K	Q_t
0	0	Q_{t-1}
1	0	1
0	1	0
1	1	$\overline{Q_{t-1}}$

2.2. Реализация JK триггера в схемотехническом редакторе Xilinx ISE DS 14.1

Схема JK триггера в программной оболочке ISE представлена на рис. 2.11.

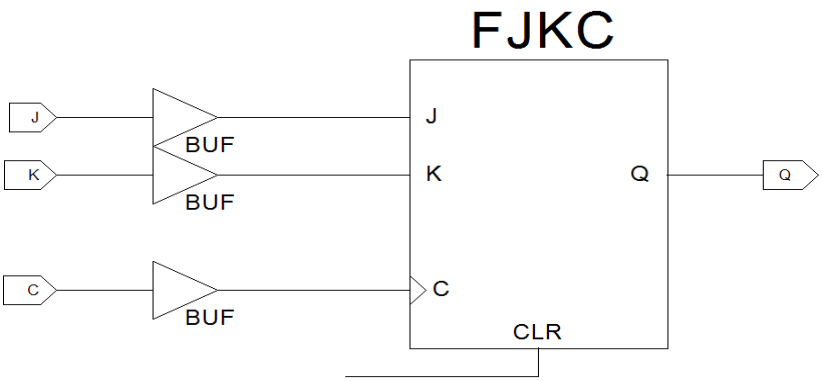


Рис. 2.11. Схема JK триггера, тактируемого по переднему фронту, в схемотехническом редакторе Xilinx ISE Design Suite 14.1

2.3. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
  …
PROCESS
   begin
J<='0';
K<='0';
C<='0';
 wait for 100 ns;
J<='1';
K<='0';
C<='0';
 …
     WAIT;
  END PROCESS;
```

2.4. Описание JK триггера на языке VHDL

library ieee;
use ieee.std_logic_1164.all;
entity jk_trigger is
  port (
  Clk, Clr, J, K : in std_logic;
  Q,notQ : out std_logic);
end jk_trigger;
 	
architecture behaviour of jk_trigger is
  signal Qtmp : std_logic := '0';
  signal notQtmp : std_logic := '1';
begin
  Q <= Qtmp;
  notQ <= notQtmp;
 	
  process (Clk, Clr)
  begin
  if Clr = '1' then
  Qtmp <= '0';
  notQtmp <= '1';
  elsif Clk'event and Clk = '1' then
  if (J = '1') and (K = '0') then
  Qtmp <= '1';
  notQtmp <= '0';
  elsif (J = '0') and (K = '1') then
  Qtmp <= '0';
  notQtmp <= '1';
  elsif (J = '1') and (K = '1') then
  Qtmp <= not Qtmp;
  notQtmp <= not notQtmp;
  end if;
  end if;
  end process;
end behaviour;
	Подключаем все объявления пакета std_logic_1164, входящего в библиотеку IEEE
	
	Определяем сигналы, которыми объект будет обмениваться
	in – входные порты
	out – выходные порты
	
	Архитектурное тело
	
	Присвоение значения сигналу **<=**
	when – когда
	else - иначе

2.5. Проверка работоспособности в симуляторе ISim

```
  …
    constant Clk_period : time := 100 ns;
  BEGIN
    uut: VHDL_JK PORT MAP (
          Clk => Clk,
          Clr => Clr,
          J => J,
          K => K,
          Q => Q,
          notQ => notQ
        );
    Clk_process :process
    begin
  Clk <= '0';
  wait for Clk_period/2;
  Clk <= '1';
  wait for Clk_period/2;
    end process;
stim_proc: process
    begin
   J<='0';
  K<='0';
   wait for 100 ns;
   J<='1';
  K<='0';
   wait for 100 ns;
  …
    wait;
    end process;
  END;
```

Приведем пример различных вариантов D триггера.

D-триггер, тактируемые задним фронтом

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity D_trigger is
    Port ( data : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           q : out  STD_LOGIC);
end D_trigger;architecture Behavioral of D_trigger is
begin
process (clk) begin
if (clk'event and clk = '0') then
q <= data;
end if;
end process;
end Behavioral;
```

D-защелка (D-триггер, тактируемый уровнем)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity zashelka is
    Port ( Data : in  STD_LOGIC;
           Clk : in  STD_LOGIC;
           Q : out  STD_LOGIC);
end zashelka;
architecture Behavioral of zashelka is
begin
process (Clk) begin
if (Clk='1') then Q<=Data;
end if;
end process;
end Behavioral;
```

D-триггер с инверсным входом асинхронной установки

```
library IEEE;
use IEEE.std_logic_1164.all;
entity dff_async_pre is
port (data, clk, preset : in std_logic;
q :out std_logic);
end dff_async_pre;
architecture behav of dff_async_pre is
begin
process (clk, preset) begin
if (preset = '0') then
q <= '1';
elsif (clk'event and clk = '1') then
q <= data;
end if;
end process;
end behav;
```

D-триггер с инверсными входами асинхронного сброса и установки

```
library IEEE;
use IEEE.std_logic_1164.all;
entity dff_async is
port (data, clk, reset, preset : in std_logic;
q :out std_logic);
end dff_async;
architecture behav of dff_async is
begin
process (clk, reset, preset) begin
if (reset = '0') then
q <= '0';
elsif (preset = '0') then
q <= '1';
elsif (clk'event and clk = '1') then
q <= data;
```

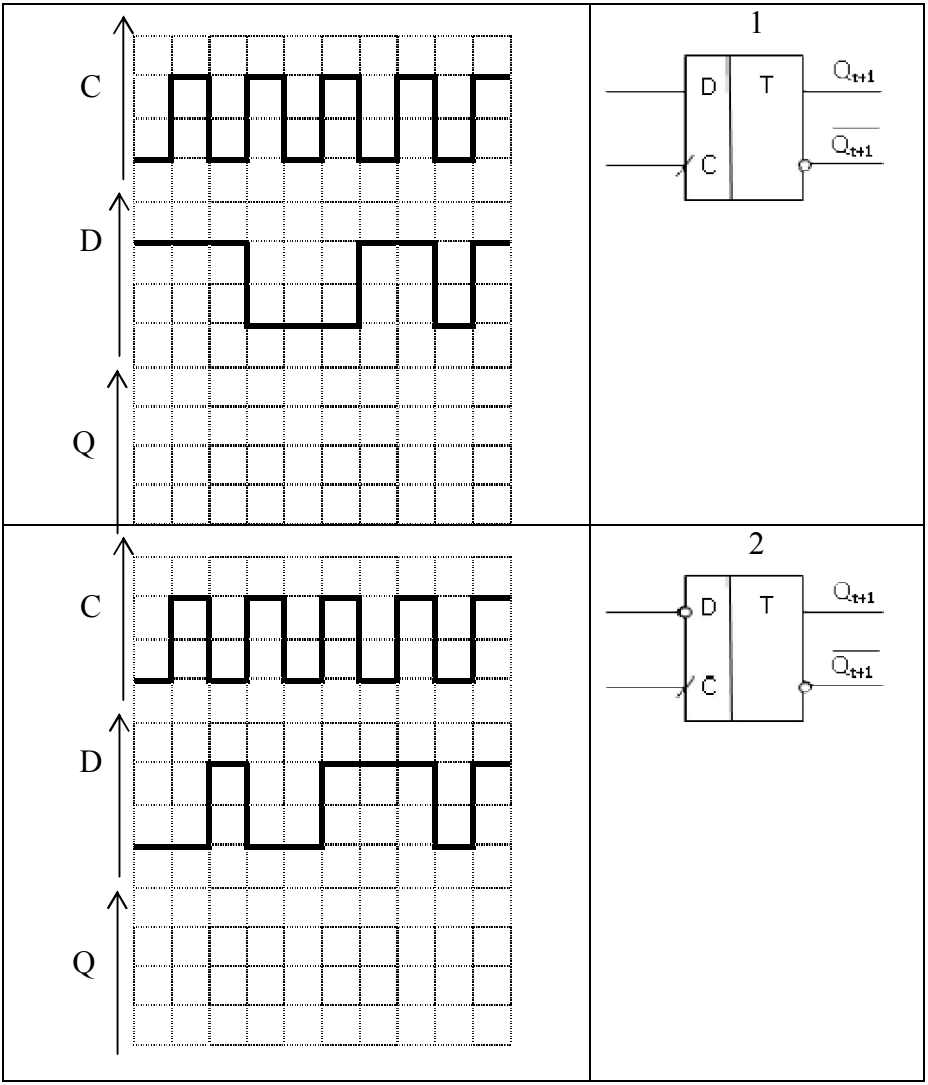
```
elsif (reset = '1' and preset = '1') then
q <= '0';
end if;
end process;
end behav;
```

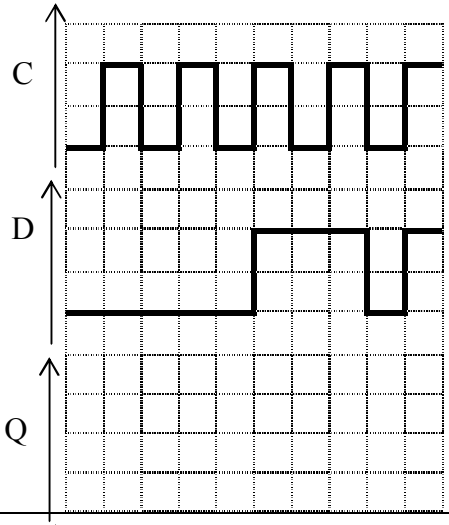
Контрольные вопросы

1. Асинхронный RS – триггер: словесное описание, таблицы истинности, переключательные функции, схемы функционирование.
2. Синхронный RS – триггер: словесное описание, таблицы истинности, переключательные функции, схемы функционирование.
3. D-триггер: словесное описание, таблицы истинности, переключательные функции, схемы функционирование.
4. Асинхронный T-триггер: словесное описание, таблицы истинности, переключательные функции, схемы функционирование.
5. Синхронный T-триггер: словесное описание, таблицы истинности, переключательные функции, схемы функционирование.
6. JK-триггер на основе D-триггера: схема, таблица истинности, переключательная функция, функционирование.
7. Преобразование JK-триггера в RS-, D-, T-триггеры и D-триггера в T-триггер.

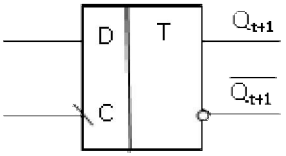
Индивидуальное задание

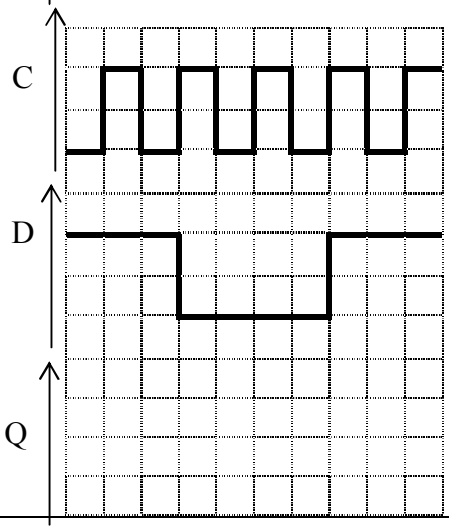
1. Для выполнения первого пункта лабораторной работы



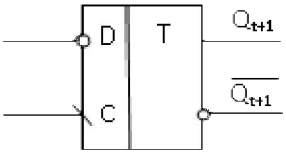


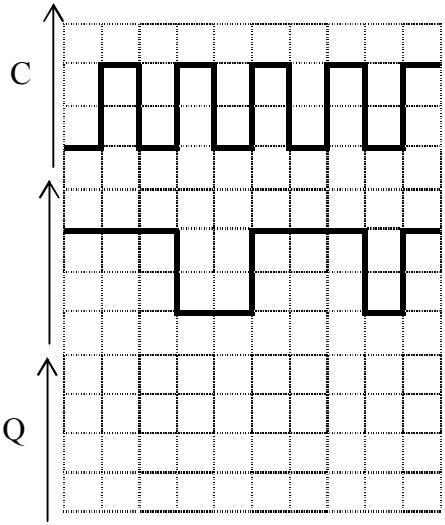
3



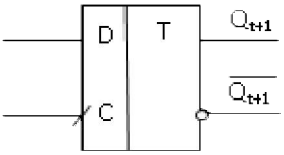


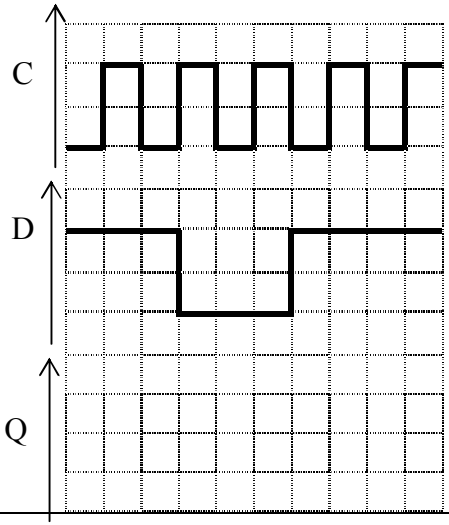
4



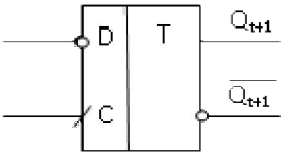


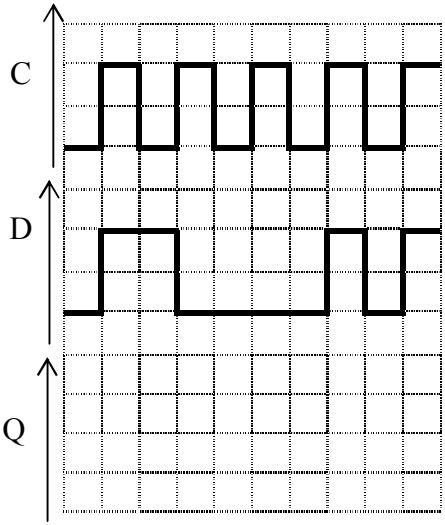
5



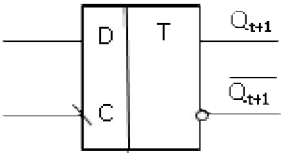


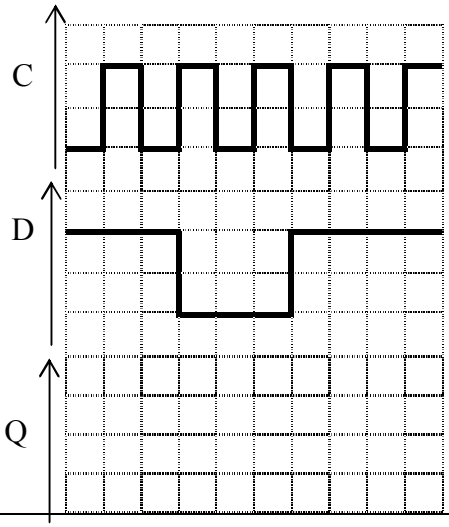
6



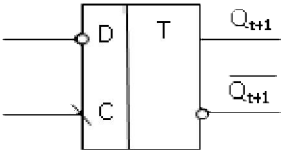


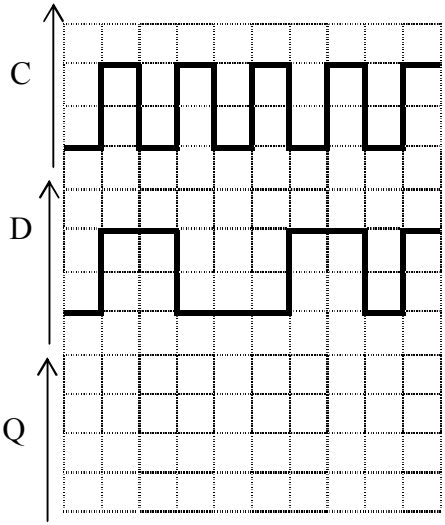
7



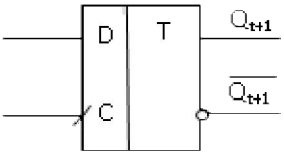


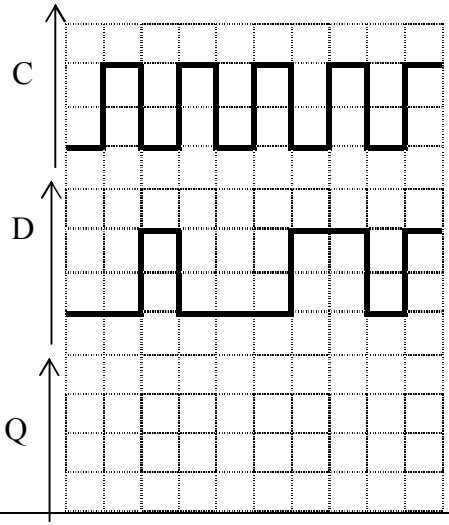
8



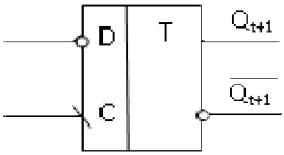


9

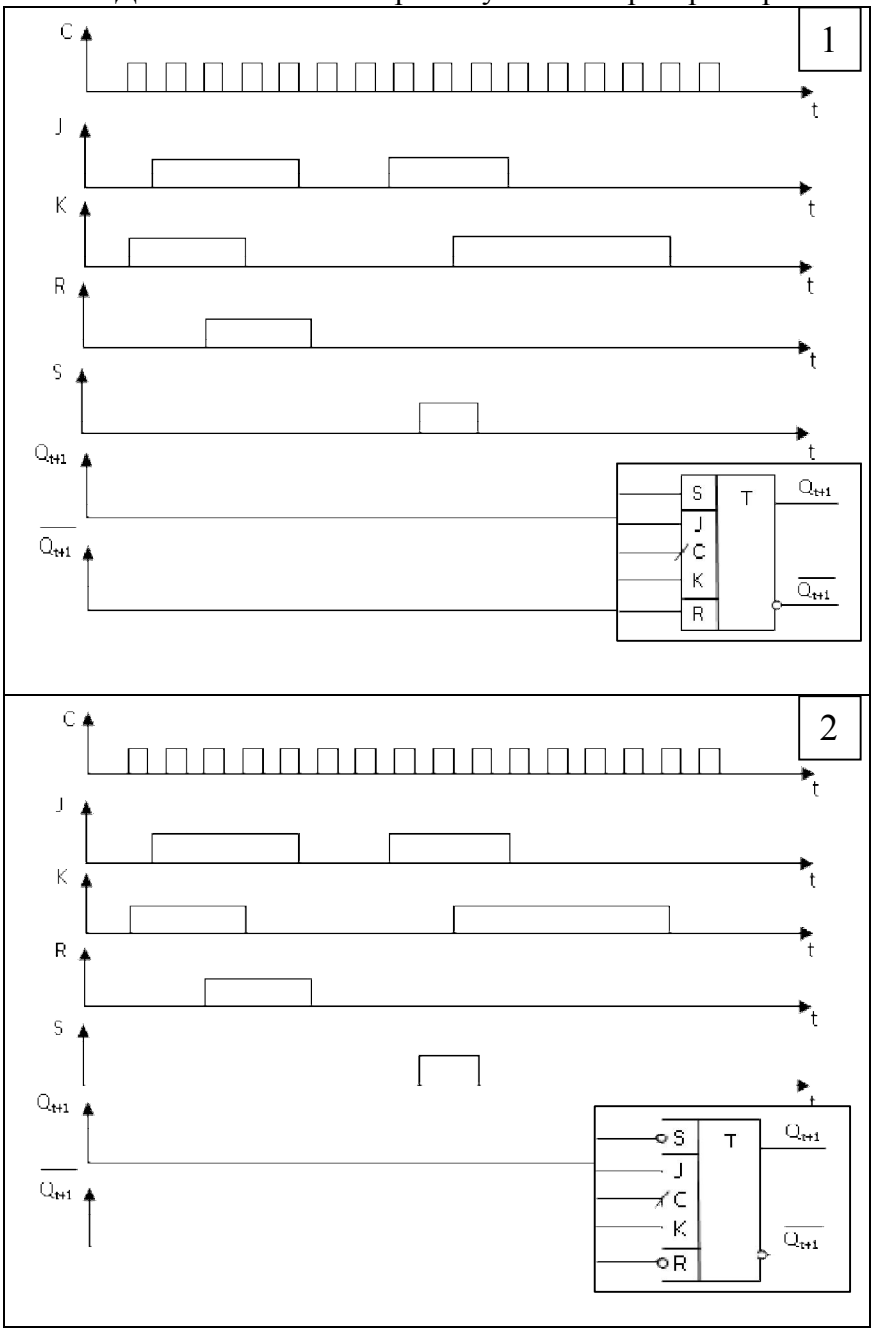


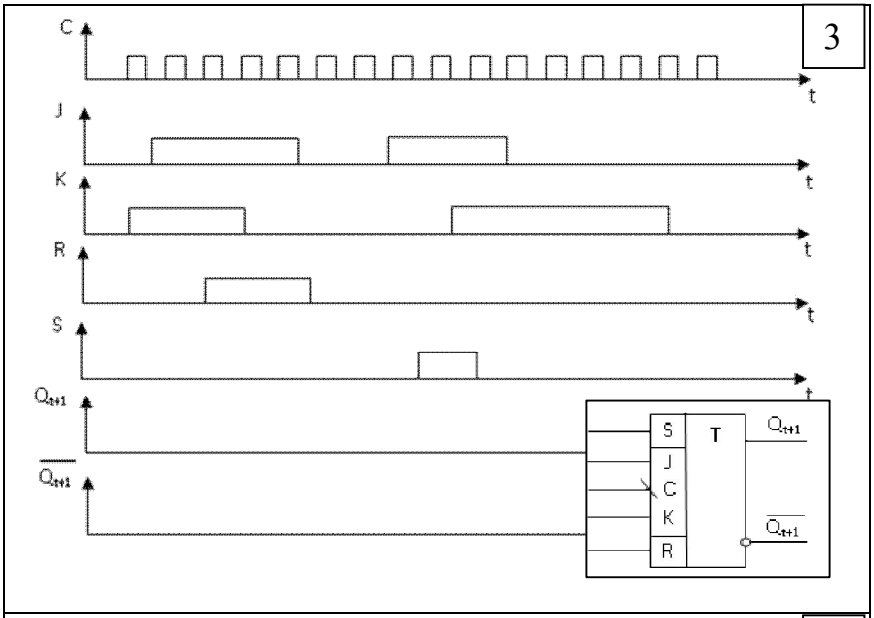


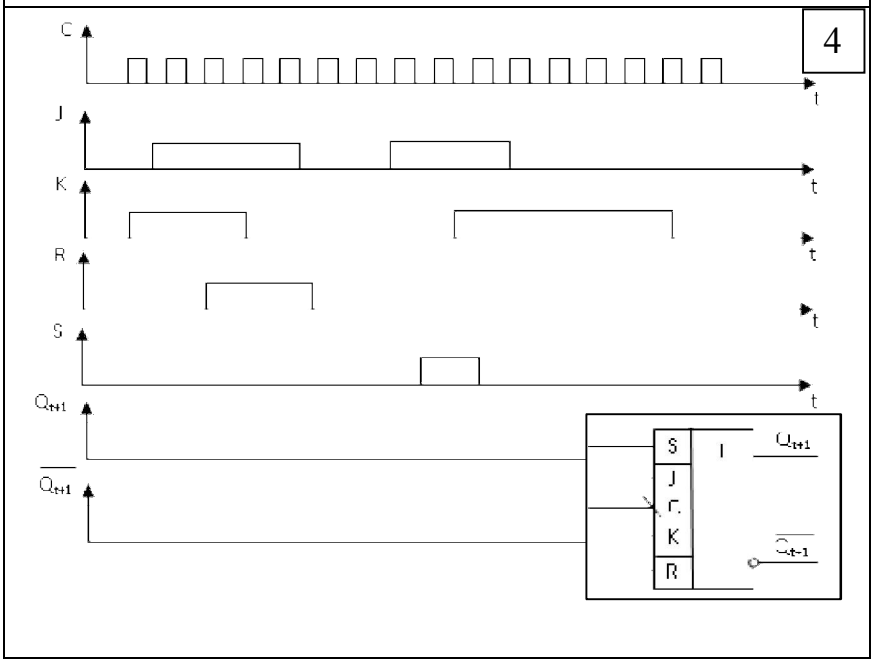
10



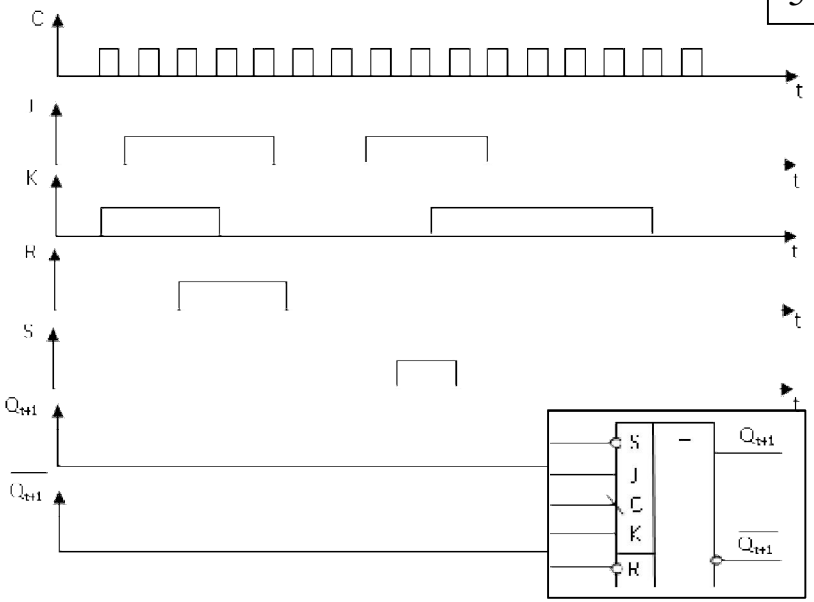
2. Для выполнения второго пункта лабораторной работы



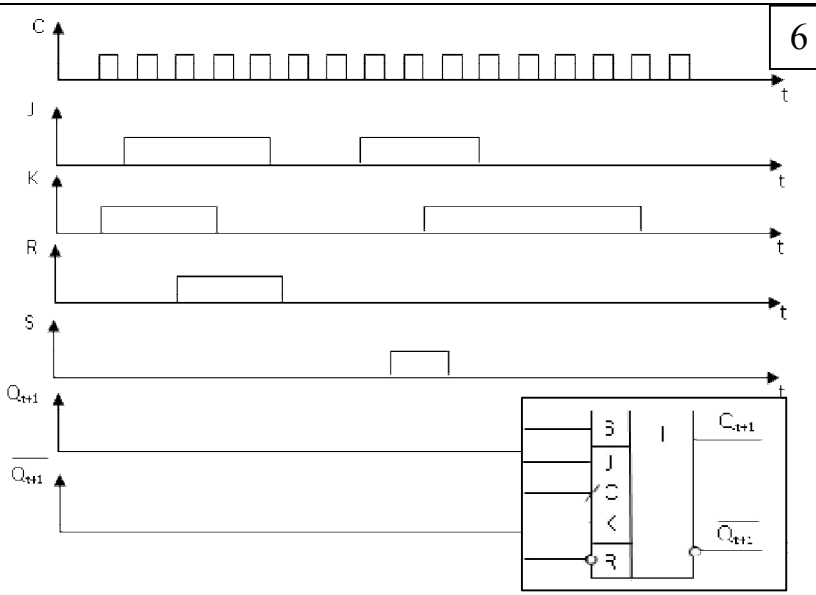

3
C
t
J
K
R
S
Q_{t+1}
$\overline{Q_{t+1}}$
S
J
C
K
R
T


4
C
t
J
K
R
S
Q_{t+1}
$\overline{Q_{t+1}}$
S
J
K
R
T

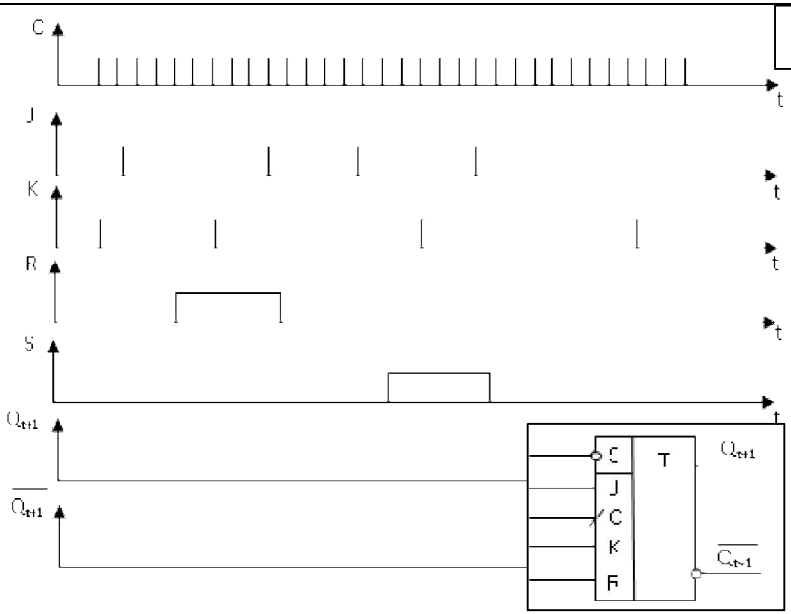
5


C
t
J
K
R
S
Q_{t+1}
$\overline{Q_{t+1}}$
S
J
C
K
R
–
Q_{t+1}
$\overline{Q_{t+1}}$

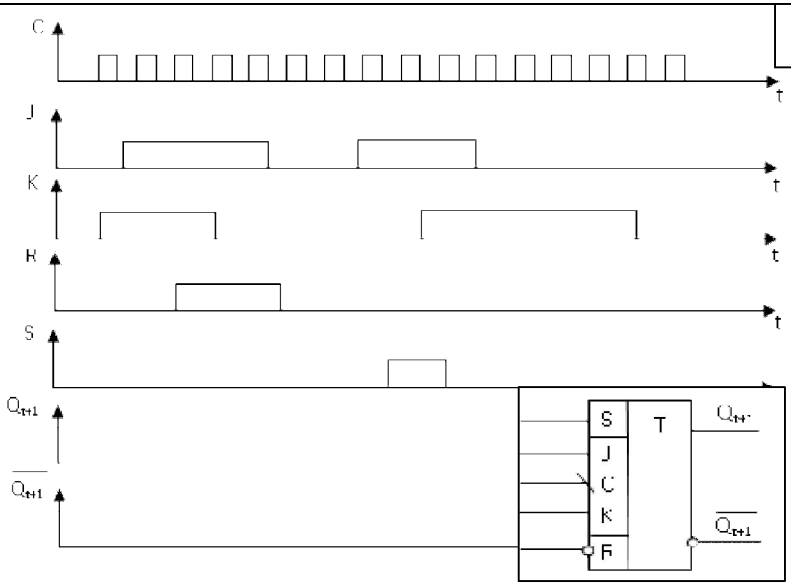
6


C
t
J
K
R
S
Q_{t+1}
$\overline{Q_{t+1}}$
S
J
C
K
R
1
C_{t+1}
$\overline{Q_{t+1}}$

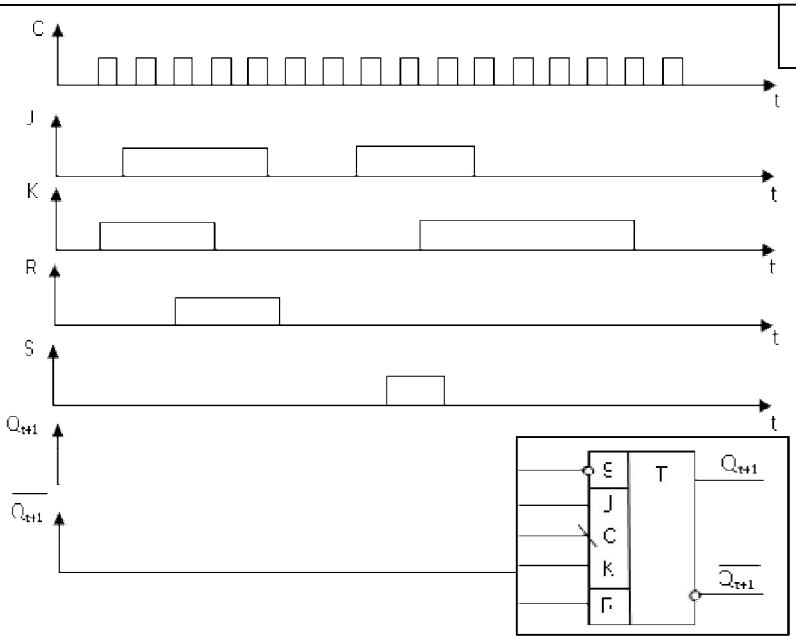
7


C
t
J
K
R
S
Q_{t+1}
$\overline{Q_{t+1}}$
C
J
C
K
R
T
Q_{t+1}
$\overline{Q_{t+1}}$

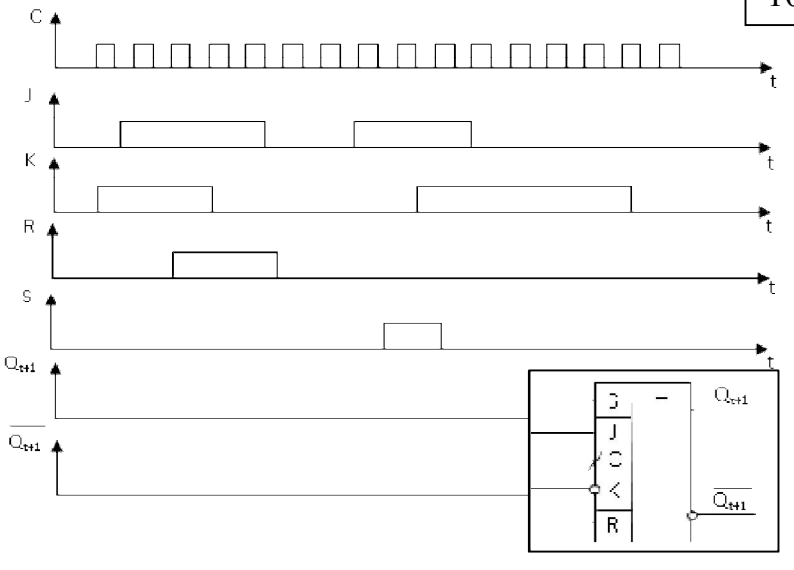
8


C
t
J
K
R
S
Q_{t+1}
$\overline{Q_{t+1}}$
S
J
C
K
R
T
Q_{t+1}
$\overline{Q_{t+1}}$

9


C
t
J
t
K
t
R
t
S
t
Q_{t+1}
$\overline{Q_{t+1}}$
T
Q_{t+1}
$\overline{Q_{t+1}}$
J
C
K
R

10


C
t
J
t
K
t
R
t
S
t
Q_{t+1}
$\overline{Q_{t+1}}$
J
C
K
R
Q_{t+1}
$\overline{Q_{t+1}}$

Лабораторная работа № 3

РАЗРАБОТКА И ИССЛЕДОВАНИЕ ДЕШИФРАТОРА И ШИФРАТОРА

Цель работы

1. Получение практических навыков в разработке и исследовании дешифраторов и шифраторов в заданном базисе.

2. Привитие навыков проведения контроля работоспособности цифровых функциональных узлов комбинационного типа.

3. Приобретение практических навыков использования системы виртуального схемотехнического моделирования Xilinx ISE Design Suite 14.1.

Содержание работы

1. Разработка и исследование шифратора в заданном базисе.

2. Разработка и исследование дешифратора в заданном базисе.

Литература

1. Бибило П.Н. Основы языка VHDL: Учебное пособие. Изд. 6-е. - М.: Книжный дом «ЛИБРОКОМ», 2014. – 328 с.

2. Бабак В. П., Корченко А. Г., Тимошенко Н. П., Филоненко С. Ф. VHDL. Справочное пособие по основам языка – М.: Издательский дом «Додэка-XXI», 2008. – 224 с.

3. Тарасов И.Е. Разработка цифровых устройств на основе ПЛИС Xilinx с применением языка VHDL. –Изд. 2-е. – М.: Горячая линия – Телеком, 2015. – 252 с.

Содержание отчета

Цель работы.

1. Разработка шифратора;

1.1 Таблица истинности в соответствии с индивидуальным заданием;

1.2 Минимизированная переключательная функция (ПФ) в заданном базисе;

1.3 Реализация ПФ в схемотехническом редакторе Xilinx ISE Design Suite 14.1;

1.4 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

1.5 Код программы на языке VHDL для описания шифратора с соответствии с индивидуальным заданием;

1.6 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

1.7 Трансляция разработанного проекта, программирование ПЛИС;

2. Разработка дешифратора;

2.1 Таблица истинности в соответствии с индивидуальным заданием;

2.2 Минимизированная ПФ в заданном базисе;

2.3 Реализация ПФ в схемотехническом редакторе Xilinx ISE Design Suite 14.1;

2.4 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

2.5 Код программы на языке VHDL для описания дешифратора с соответствии с индивидуальным заданием;

2.6 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

2.7 Трансляция разработанного проекта, программирование ПЛИС.

3. Выводы.

Теоретическая часть

Шифратором называется комбинационное цифровое устройство с m входами и n выходами, преобразующий сигнал логической единицы на одном из входов в n-разрядный параллельный двоичный код.

Максимальное число входов m_{max} определяется числом возможных кодовых комбинаций и составляет 2^n. Условное графическое обозначение шифратора показано на рис. 2.12. Входные шины нумеруются от 0 до m-1, а на выходных шинах обозначается вес двоичного разряда 2^0, 2^1, 2^2, 2^3, … 2^{n-1}. Как следует из определения шифратора, только небольшая часть наборов входных переменных таблицы истинности соответствует кодовым комбинациям на выходе, остальные 2^n - m наборов являются запрещенными. Поэтому оператор, выполняемый шифратором, удобно задавать сокращенной таблицей истинности, содержащей только m строк.

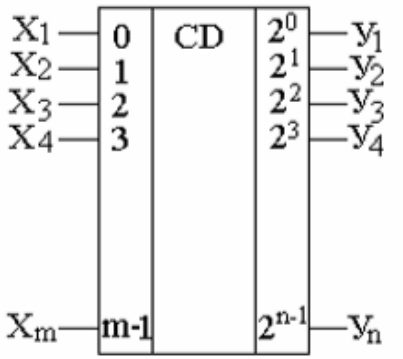


Рис. 2.12. Условное графическое обозначение шифратора

Дешифратором называется комбинационное цифровое устройство с m входами и n выходами, преобразующий m-разрядный параллельный код, поступающий на входы, в сигнал логической единицы на одном из выходов.

Каждому выходу соответствует своя кодовая комбинация на входах. Условное обозначение дешифратора показано на

рис. 2.13. Входы обозначаются весами двоичных разрядов от 2^0 до 2^m - 1, а выходы нумеруются от 0 до (n-1). Так как на m входах может быть 2^m наборов входных переменных, то максимальное число выходов равно $n_{max} = 2^m$. Если используются все выходы, дешифратор называется полным, если же число выходов меньше 2^m – неполным.

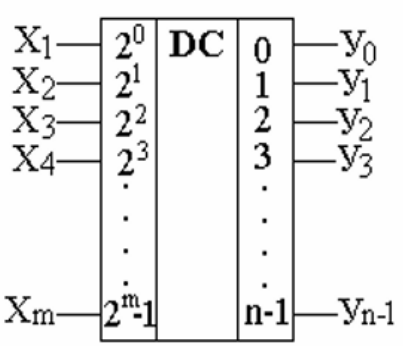


Рис. 2.13. Условное графическое обозначение дешифратора

Порядок выполнения работы

1. Разработка шифратора
1.1. Таблица истинности шифратора 5 х 3

Рассмотрим пример построения шифратора 5 х 3 на базе логических элементов «3ИЛИ-НЕ». Приведем таблицу истинности шифратора (табл. 2.13).

Таблица 2.13

Таблица истинности шифратора

№	X5	X4	X3	X2	X1	**У2**	**У1**	**У0**
0	0	0	0	0	1	0	0	1
1	0	0	0	1	0	0	1	0
2	0	0	1	0	0	0	1	1
3	0	1	0	0	0	1	0	0
4	1	0	0	0	0	1	0	1

1.2. Минимизированная переключательная функция шифратора 5 х 3 в базисе «3ИЛИ-НЕ»

Младший разряд выходного двоичного кода (Y0) устанавливается в логическую единицу при высоком уровне на хотя бы одном из нечетных входах (X1, X3, X5) $Y_0 = X_1 + X_3 + X_5$.

Первый разряд выходного двоичного кода (Y1) устанавливается в логическую единицу при высоком уровне на входах X2 или X3, т.е. $Y_1 = X_2 + X_3$.

Второй разряд выходного двоичного кода (Y2) устанавливается в логическую единицу при высоком уровне на входах X4 или X5, т.е. $Y_2 = X_4 + X_5$.

Переключательная функция в базисе «3ИЛИ-НЕ» имеет вид:

$$y_0 = \overline{\overline{x_1 + x_3 + x_5}},$$
$$y_1 = \overline{\overline{x_2 + x_3}},$$
$$y_2 = \overline{\overline{x_4 + x_5}}.$$

1.3. Реализация ПФ в схемотехническом редакторе Xilinx ISE Design Suite 14.1

Схема шифратора в программной оболочке ISE представлена на рис. 2.14.

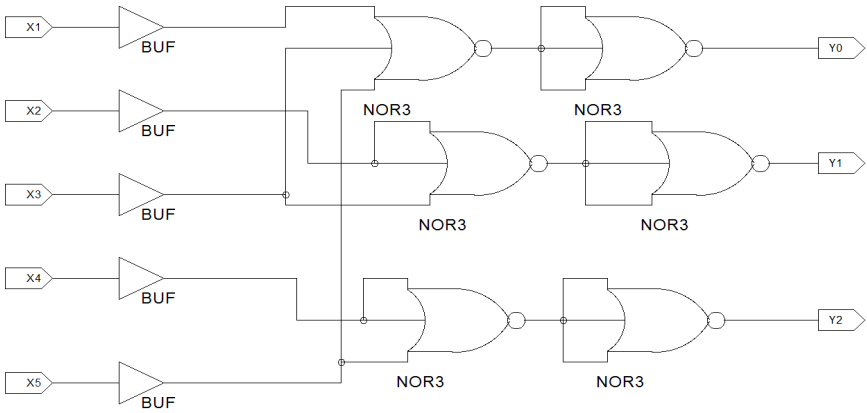


Рис. 2.14. Схема шифратора 5 х 3 в базисе «3ИЛИ-НЕ»

1.4. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
   …
PROCESS
  BEGIN
  X1<='1';   //присваиваем переменной X1 значение 1;
  X2<='0';   //присваиваем переменной X2 значение 0;
  …
  X5<='0';  //присваиваем переменной X5 значение 0;
 wait for 50 ns;
     …        //добавляем необходимые значения во временном
               промежутке с шагом 50 ns
  WAIT;
   END PROCESS;
```

1.5. Код программы на языке VHDL для описания шифратора 5x3

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity shifrator is
Port (Ain : in
STD_LOGIC_VECTOR (4 downto 0);
Yout : out
STD_LOGIC_VECTOR(2 downto 0));
end shifrator;
architecture Behavioral of deshifrator is
begin
PROCESS (Ain)
BEGIN
case Ain is
when "00001"=>Yout<="001";
when "00010"=>Yout<="010";
when "00100"=>Yout<="011";
when "01000"=>Yout<="100";
when others=>Yout<="101";
end case;
END PROCESS;
end Behavioral;
	Подключаем все объявления пакета STD_LOGIC_1164, входящего в библиотеку IEEE
	
	Определяем сигналы, которыми объект будет обмениваться
	In – входные порты, Ain - пятиразрядный вектор типа std_logic_vector
	Out – выходные порты, Yout - трехразрядный вектор типа std_logic_vector
	std_logic - это просто провод.
	std_logic_vector - группа проводов
	
	case X is
	when 1 => <оператор>
	…
	when others => <оператор>
	end case;
	Ключевое слово others определяет операторы, которые исполняются, если значение ключевого выражения не совпадает ни с одним вариантом и не выходит в объявленные диапазоны

1.6. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
process
  begin
    Ain<="00001";
    wait for 100 ns;
    …
  end process;
```

Таблица 2.14

Индивидуальное задание

№	Тип шифратора	Базис
1	Шифратор 5 х 3	2 И-НЕ
2	Шифратор 4 х 2	2 И-НЕ
3	Шифратор 8 х 3	2 И-НЕ
4	Шифратор 6 х 3	2 И-НЕ
5	Шифратор 5 х 3	2 ИЛИ-НЕ
6	Шифратор 4 х 2	2 ИЛИ-НЕ
7	Шифратор 8 х 3	2 ИЛИ-НЕ
8	Шифратор 6 х 3	2 ИЛИ-НЕ
9	Шифратор 5 х 3	3 И-НЕ
10	Шифратор 4 х 2	3 И-НЕ
11	Шифратор 8 х 3	3 И-НЕ
12	Шифратор 6 х 3	3 И-НЕ
13	Шифратор 5 х 3	3 ИЛИ-НЕ
14	Шифратор 4 х 2	3 ИЛИ-НЕ
15	Шифратор 8 х 3	3 ИЛИ-НЕ
16	Шифратор 6 х 3	3 ИЛИ-НЕ

2. Разработка дешифратора

2.1. Таблица истинности неполного дешифратора 3х5

Рассмотрим пример построения неполного дешифратора 3 х 5 на базе логических элементов «3ИЛИ-НЕ». Приведем таблицу истинности дешифратора (табл. 2.15).

Таблица 2.15

Таблица истинности неполного дешифратора 3х5

№	X3	X2	X1	**У5**	**У4**	**У3**	**У2**	**У1**
0	0	0	1	0	0	0	0	1
1	0	1	0	0	0	0	1	0
2	0	1	1	0	0	1	0	0
3	1	0	0	0	1	0	0	0
4	1	0	1	1	0	0	0	0

2.2. Минимизированная переключательная функция дешифратора 3 х 5 в базисе «3ИЛИ-НЕ»

Дешифратор можно описать следующими ПФ в СДНФ:

$$Y_1 = \overline{X_3} \cdot \overline{X_2} \cdot X_1,$$
$$Y_2 = \overline{X_3} \cdot X_2 \cdot \overline{X_1},$$
$$Y_3 = \overline{X_3} \cdot X_2 \cdot X_1,$$
$$Y_4 = X_3 \cdot \overline{X_2} \cdot \overline{X_1},$$
$$Y_5 = X_3 \cdot \overline{X_2} \cdot X_1.$$

Запишем ПФ в базисе «3ИЛИ-НЕ».

$$Y_1 = \overline{\overline{X_3 + X_2 + \overline{X_1}}},$$
$$Y_2 = \overline{\overline{X_3 + \overline{X_2} + X_1}},$$
$$Y_3 = \overline{\overline{X_3 + \overline{X_2} + \overline{X_1}}},$$
$$Y_4 = \overline{\overline{\overline{X_3} + X_2 + X_1}},$$
$$Y_1 = \overline{\overline{\overline{X_3} + X_2 + \overline{X_1}}}.$$

2.3. Реализация ПФ в схемотехническом редакторе Xilinx ISE Design Suite 14.1

Схема шифратора в программной оболочке ISE представлена на рис. 2.15.

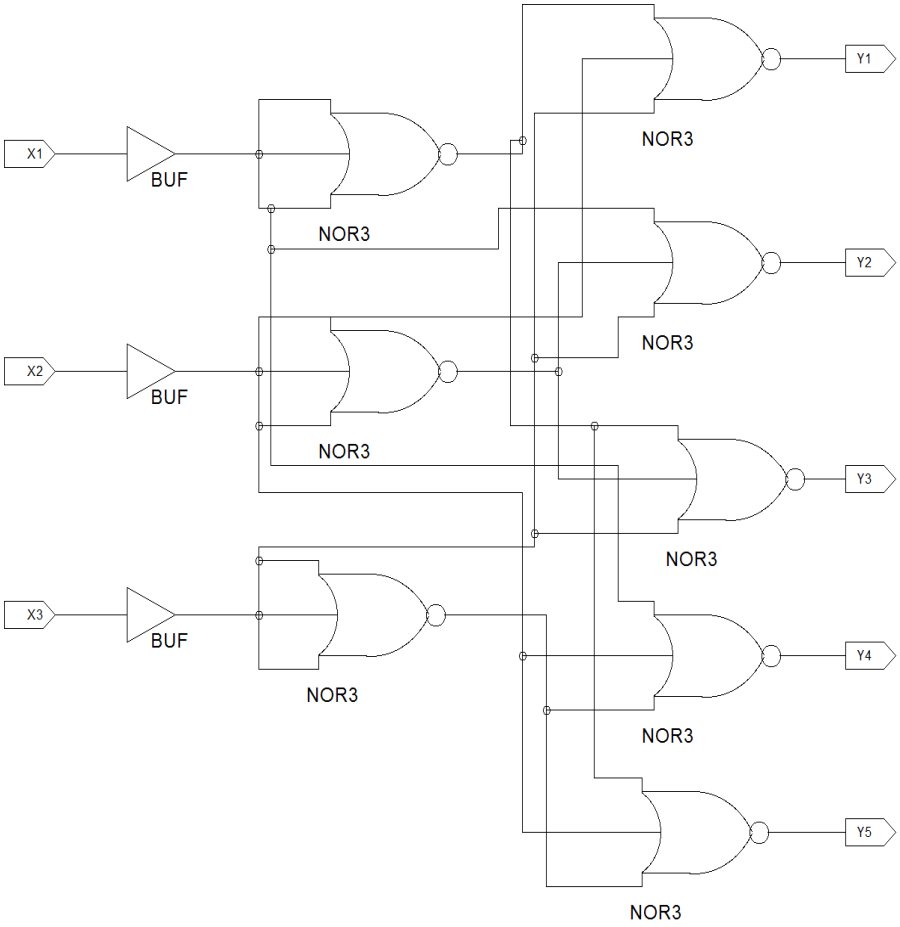


Рис. 2.15. Схема неполного дешифратора 3 х 5 в базисе «ЗИЛИ-НЕ»

2.4. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
   …
PROCESS
  BEGIN
X1<='1';
X2<='0';
X3<='0';
wait for 50 ns;
    …       //добавляем необходимые значения во временном
              промежутке с шагом 50 ns
 WAIT;
  END PROCESS;
  …
```

2.5. Код программы на языке VHDL для описания дешифратора 3x5

library IEEE; use IEEE.STD_LOGIC_1164.ALL; entity DESH_VHDL_3x5 is	Подключаем все объявления пакета STD_LOGIC_1164, входящего в библиотеку IEEE
Port (Ain : in STD_LOGIC_VECTOR (2 downto 0); Yout : out STD_LOGIC_VECTOR(4 downto 0)); end DESH_VHDL_3x5; architecture Behavioral of DESH_VHDL_3x5 is begin PROCESS (Ain)	Определяем сигналы, которыми объект будет обмениваться In – входные порты, Ain – трехразрядный вектор типа std_logic_vector Out – выходные порты, Yout - пятиразрядный вектор типа std_logic_vector *std_logic - это просто провод.* *std_logic_vector - группа*

begin case Ain is when "001"=>Yout<="00001"; when "010"=>Yout<="00010"; when "011"=>Yout<="00100"; when "100"=>Yout<="01000"; when others=>Yout<="10000"; end case; END PROCESS; end Behavioral;	*проводов* case X is when 1 => <оператор> … when others => <оператор> end case; Ключевое слово others определяет операторы, которые исполняются, если значение ключевого выражения не совпадает ни с одним вариантом и не выходит в объявленные диапазоны

2.6. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
process
  begin
    Ain<="001";
    wait for 100 ns;
    …
  end process;
```

Таблица 2.16

Индивидуальное задание

№	Тип дешифратора	Базис
1	Дешифратор 5 х 3	2 И-НЕ
2	Дешифратор 4 х 2	2 И-НЕ
3	Дешифратор 8 х 3	2 И-НЕ
4	Дешифратор 6 х 3	2 И-НЕ
5	Дешифратор 5 х 3	2 ИЛИ-НЕ
6	Дешифратор 4 х 2	2 ИЛИ-НЕ
7	Дешифратор 8 х 3	2 ИЛИ-НЕ
8	Дешифратор 6 х 3	2 ИЛИ-НЕ
9	Дешифратор 5 х 3	3 И-НЕ
10	Дешифратор 4 х 2	3 И-НЕ
11	Дешифратор 8 х 3	3 И-НЕ
12	Дешифратор 6 х 3	3 И-НЕ
13	Дешифратор 5 х 3	3 ИЛИ-НЕ
14	Дешифратор 4 х 2	3 ИЛИ-НЕ
15	Дешифратор 8 х 3	3 ИЛИ-НЕ
16	Дешифратор 6 х 3	3 ИЛИ-НЕ

Контрольные вопросы

1.Назначение шифратора: схемы и функционирование;
2.Назначение дешифратора: схемы и функционирование;
3.Принцип построения одноступенчатого дешифратора;
4.Принцип построения двухступенчатого дешифратора;
5.Реализация дешифратора на языке VHDL;
6.Реализация шифратора на языке VHDL;
7.Определение полного дешифратора;
8.Определение полного шифратора.

Лабораторная работа № 4

РАЗРАБОТКА И ИССЛЕДОВАНИЕ МУЛЬТИПЛЕКСОРА И ДЕМУЛЬТИПЛЕКСОРА

Цель работы

1. Получение практических навыков в разработке и исследовании мультиплексоров и демультиплексоров в заданном базисе.

2. Привитие навыков проведения контроля работоспособности цифровых функциональных узлов комбинационного типа.

3. Приобретение практических навыков использования системы виртуального схемотехнического моделирования Xilinx ISE Design Suite 14.1.

Содержание работы

1. Разработка и исследование мультиплексора в заданном базисе.

2. Разработка и исследование демультиплексора в заданном базисе.

Литература

1. Бибило П.Н. Основы языка VHDL: Учебное пособие. Изд. 6-е. - М.: Книжный дом «ЛИБРОКОМ», 2014. – 328 с.

2. Бабак В. П., Корченко А. Г., Тимошенко Н. П., Филоненко С. Ф. VHDL. Справочное пособие по основам языка – М.: Издательский дом «Додэка-XXI», 2008. – 224 с.

3. Тарасов И.Е. Разработка цифровых устройств на основе ПЛИС Xilinx с применением языка VHDL. –Изд. 2-е. – М.: Горячая линия – Телеком, 2015. – 252 с.

Содержание отчета

Цель работы.

1. Разработка мультиплексора;

1.1 Таблица истинности в соответствии с индивидуальным заданием;

1.2 Минимизированная переключательная функция (ПФ) в заданном базисе;

1.3 Реализация ПФ в схемотехническом редакторе Xilinx ISE Design Suite 14.1;

1.4 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

1.5 Код программы на языке VHDL для описания мультиплексора в соответствии с индивидуальным заданием;

1.6 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

1.7 Трансляция разработанного проекта, программирование ПЛИС;

2. Разработка демультиплексора;

2.1 Таблица истинности в соответствии с индивидуальным заданием;

2.2 Минимизированная ПФ в заданном базисе;

2.3 Реализация ПФ в схемотехническом редакторе Xilinx ISE Design Suite 14.1;

2.4 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

2.5 Код программы на языке VHDL для описания демультиплексора с соответствии с индивидуальным заданием;

2.6 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

2.7 Трансляция разработанного проекта, программирование ПЛИС.

3. Выводы.

Теоретическая часть

Мультиплексор – это функциональный узел комбинационного типа, предназначенный для коммутации цифровых сигналов, поступающих по нескольким информационным входам, на один выход. При этом каждый информационный входе имеет свой адрес, который задается n-разрядным цифровым кодом. Количество информационных входов равно $N = 2^n$. На рис. 2.16 изображено условное графическое обозначение 4–х входового мультиплексора с управлением двоичным кодом. Входы A0 и A1 являются управляющими входами мультиплексора, определяющими адрес информационного входного сигнала, который будет соединён с выходным выводом мультиплексора Y. Информационные входные сигналы обозначены: X0, X1, X2 и X3.

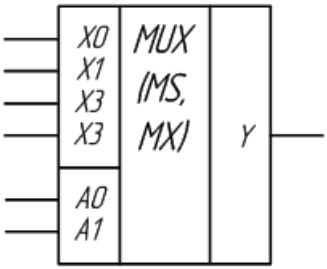


Рис. 2.16. Условное графическое обозначение мультиплексора MS 4:1

Демультиплексор – это функциональный узел комбинационного типа, предназначенный для переключения сигнала с одного информационного входа на один из нескольких информационных выходов. Демультиплексоры выполняют функцию, обратную мультиплексорам, т.е. один входной сигнал распределяют по нескольким выходам. При этом каждый выход имеет свой адрес, который задается n-разрядным цифровым кодом. Количество выходов равно $N = 2^n$. На рис. 2.17 изображено условное графическое обозначение мультиплексора с управлением

двоичным кодом. Входы A0 и A1 являются управляющими входами мультиплексора, определяющими адрес информационного выходного сигнала, который будет соединён с входным выводом мультиплексора X.

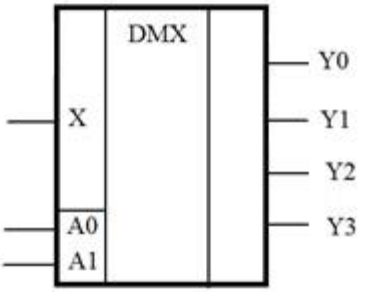


Рис. 2.17. Условное графическое обозначение
демультиплексора DMX 1:4

Порядок выполнения работы

1. Разработка мультиплексора
1.1. Таблица истинности мультиплексора 3:1

Рассмотрим пример построения мультиплексора 3:1 на базе логических элементов «3И-НЕ». Определяем разрядность двоичного кода, необходимого для коммутации заданного числа каналов n = int ($\log_2 3$), где int – ближайшее большее целое число; n = 2. Приведем таблицу истинности шифратора (табл. 2.17).

Таблица 2.17

Таблица истинности мультиплексора 3:1

A_1	A_0	Y
0	0	X0
0	1	X1
1	0	X2
1	1	*

1.2. Минимизированная переключательная функция (ПФ) в базисе «3И-НЕ»

Переключательная функция (ПФ) в СДНФ:

$$y = X_0 \cdot \overline{A_1} \cdot \overline{A_0} + X_1 \cdot \overline{A_1} \cdot A_0 + X_2 \cdot A_1 \cdot \overline{A_0}.$$

Преобразованная ПФ в базисе «3И – НЕ»:

$$y = \overline{\overline{X_0 \cdot \overline{A_1} \cdot \overline{A_0}} \cdot \overline{X_1 \cdot \overline{A_1} \cdot A_0} \cdot \overline{X_2 \cdot A_1 \cdot \overline{A_0}}}.$$

1.3. Реализация ПФ в схемотехническом редакторе Xilinx ISE Design Suite 14.1

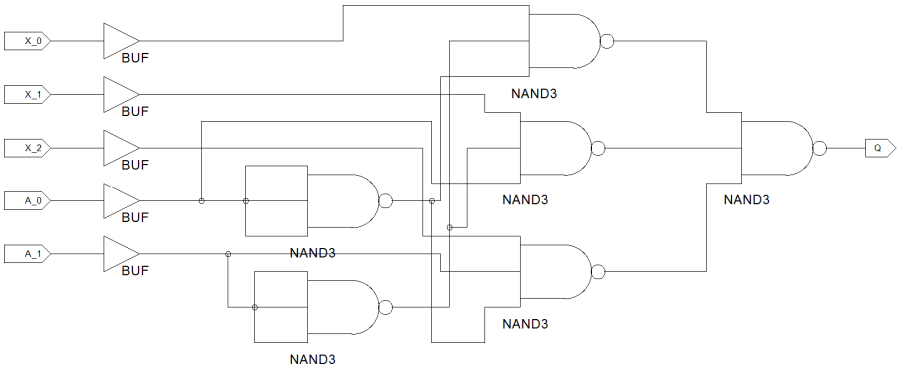


Рис. 2.18. Схема мультиплексора 3:1 в базисе «3И-НЕ»

1.4. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
   …
PROCESS
  BEGIN
X_0<='0';
X_1<='1';
X_2<='1';
A_0<='0';
A_1<='0';
wait for 50 ns;
  …       //добавляем необходимые значения во временном
              промежутке с шагом 50 ns
  WAIT;
  END PROCESS;
```

1.5. Код программы на языке VHDL для описания мультиплексора

library IEEE; use IEEE.STD_LOGIC_1164.ALL;	Подключаем все объявления пакета STD_LOGIC_1164, входящего в библиотеку IEEE
entity mux is Port (A : in STD_LOGIC_VECTOR (1 downto 0); X0 : in STD_LOGIC; X1 : in STD_LOGIC; X2 : in STD_LOGIC; F : out STD_LOGIC); end mux;	Определяем сигналы, которыми объект будет обмениваться In – входные порты, A - двухразрядный вектор типа std_logic Out – выходные порты

architecture Behavioral of mux is
begin
Process (A, X0, X1, X2)
Begin
if A="00" then F<=X0;
Elsif A="01" then F<=X1;
Elsif A="10" then F<=X2;
End if;
End process;
end Behavioral;
	if (условие) then
	// что делать, если условие верно
	else
	// что делать, если условие неверно
	
	Elsif – иначе если

1.6. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
process
  begin
A<="00";
X0<='0';
X1<='1';
X2<='1';
  wait for 100 ns;
    …
  end process;
```

Таблица 2.19

Индивидуальное задание

№	Индивидуальное задание	Заданный базис
1	Мультиплексор 3:1	2 И-НЕ
2	Мультиплексор 4:1	2 И-НЕ
3	Мультиплексор 6:1	2 И-НЕ
4	Мультиплексор 8:1	2 И-НЕ
5	Мультиплексор 3:1	2 ИЛИ-НЕ
6	Мультиплексор 4:1	2 ИЛИ-НЕ
7	Мультиплексор 6:1	2 ИЛИ-НЕ
8	Мультиплексор 8:1	2 ИЛИ-НЕ
9	Мультиплексор 3:1	3 И-НЕ
10	Мультиплексор 4:1	3 И-НЕ
11	Мультиплексор 6:1	3 И-НЕ
12	Мультиплексор 8:1	3 И-НЕ
13	Мультиплексор 3:1	3 ИЛИ-НЕ
14	Мультиплексор 4:1	3 ИЛИ-НЕ
15	Мультиплексор 6:1	3 ИЛИ-НЕ
16	Мультиплексор 8:1	3 ИЛИ-НЕ
17	Мультиплексор 4:1	2 ИЛИ - НЕ

2. Разработка демультиплексора

2.1. Таблица истинности демультиплексора 1:3

Таблица 2.20

Таблица истинности демультиплексора 1:3

A_1	A_0	Y_2	Y_1	Y_0
0	0	*	*	X
0	1	*	X	*
1	0	X	*	*
1	1	*	*	*

* - 0 или 1.

2.2. Минимизированная ПФ в базисе 3И-НЕ

$$Y_0 = \overline{\overline{X \cdot \overline{A_1} \cdot \overline{A_0}}},$$
$$Y_1 = \overline{\overline{X \cdot \overline{A_1} \cdot A_0}},$$
$$Y_2 = \overline{\overline{X \cdot A_1 \cdot \overline{A_0}}}.$$

2.3. Реализация ПФ в схемотехническом редакторе Xilinx ISE Design Suite 14.1

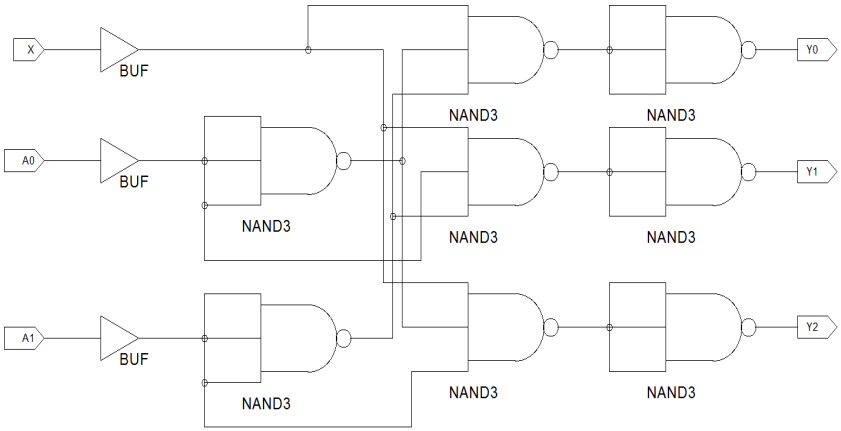


Рис. 2.19. Схема демультиплексора 1:3 в базисе «3И-НЕ»

2.4. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
PROCESS
  BEGIN
X<='0';
A0<='0';
A1<='0';
wait for 50 ns;
…
 WAIT;
  END PROCESS;
```

2.5. Код программы на языке VHDL для описания демультиплексора 1:3

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity demux is
   Port ( A : in  STD_LOGIC_VECTOR (1 downto 0);
          Y0 : out  STD_LOGIC;
          Y1 : out  STD_LOGIC;
          Y2 : out  STD_LOGIC;
          X : in  STD_LOGIC);
end demux;
architecture Behavioral of demux is
begin
Process (A, X)
Begin
if A="00" then Y0<=X;
Elsif  A="01" then Y1<=X;
Elsif  A="10" then Y2<=X;
End if;
End process;
end Behavioral;
```

2.6. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
process
  begin
A<="00";
X<='0';
  wait for 100 ns;
    …
  end process;
```

Таблица 2.22

Индивидуальное задание

№	Индивидуальное задание	Заданный базис
1	Демультиплексор 1:3	2 И-НЕ
2	Демультиплексор 1:4	2 И-НЕ
3	Демультиплексор 1:6	2 И-НЕ
4	Демультиплексор 1:8	2 И-НЕ
5	Демультиплексор 1:10	2 ИЛИ-НЕ
6	Демультиплексор 1:12	2 ИЛИ-НЕ
7	Демультиплексор 1:14	2 ИЛИ-НЕ
8	Демультиплексор 1:16	2 ИЛИ-НЕ
9	Демультиплексор 1:3	3 И-НЕ
10	Демультиплексор 1:4	3 И-НЕ
11	Демультиплексор 1:6	3 И-НЕ
12	Демультиплексор 1:8	3 И-НЕ
13	Демультиплексор 1:10	3 ИЛИ-НЕ
14	Демультиплексор 1:12	3 ИЛИ-НЕ
15	Демультиплексор 1:14	3 ИЛИ-НЕ
16	Демультиплексор 1:16	3 ИЛИ-НЕ
17	Демультиплексор 1:3	2 ИЛИ - НЕ

Контрольные вопросы

1. Дайте определение мультиплексора и демультиплексора.

2. Как определить количество каналов в мультиплексоре (демультиплексоре).

3. В чем суть каскадирования мультиплексоров? Объясните как на основе ИС мультиплексоров “8-1” спроектировать мультиплексор на 16, 32, и т.д. входов.

4. На основе ИС мультиплексора “8-1” спроектируйте схему, реализующую логическую функцию четности трехразрядного слова (четности числа единиц в трехразрядном слове).

5. На основе ИС мультиплексора “8-1” спроектируйте схему, реализующую логическую функцию нечетности трехразрядного слова.

6. Объясните, как с помощью демультиплексора можно осуществить преобразование последовательного кода в параллельный.

7. Объясните, как с помощью мультиплексора можно осуществить преобразование параллельного кода в последовательный.

8. Данные от одного из четырех источников должны последовательно передаваться по одной линии одному из трех приемников. Спроектируйте схемы и объясните работу ЦУ передающей и приемной сторон, обеспечивающих такую возможность.

Лабораторная работа № 5

РАЗРАБОТКА И ИССЛЕДОВАНИЕ РЕГИСТРОВ

Цель работы

1. Получение практических навыков в разработке и исследовании регистров.

2. Привитие навыков проведения контроля работоспособности цифровых функциональных узлов комбинационного типа.

3. Приобретение практических навыков использования системы виртуального схемотехнического моделирования Xilinx ISE Design Suite 14.1.

Содержание работы

1. Разработка и исследование сдвигового регистра.

2. Разработка и исследование параллельного регистра.

Литература

1. Бибило П.Н. Основы языка VHDL: Учебное пособие. Изд. 6-е. - М.: Книжный дом «ЛИБРОКОМ», 2014. – 328 с.

2. Бабак В. П., Корченко А. Г., Тимошенко Н. П., Филоненко С. Ф. VHDL. Справочное пособие по основам языка – М.: Издательский дом «Додэка-XXI», 2008. – 224 с.

3. Тарасов И.Е. Разработка цифровых устройств на основе ПЛИС Xilinx с применением языка VHDL. –Изд. 2-е. – М.: Горячая линия – Телеком, 2015. – 252 с.

Содержание отчета

Цель работы.

1. Разработка последовательного регистра;

1.1 Реализация сдвигового регистра в схемотехническом редакторе Xilinx ISE Design Suite 14.1;

1.2 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

1.3 Код программы на языке VHDL для описания последовательного регистра в соответствии с индивидуальным заданием;

1.4 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

1.5 Трансляция разработанного проекта, программирование ПЛИС;

2. Разработка параллельного регистра;

2.1 Реализация параллельного регистра в схемотехническом редакторе Xilinx ISE Design Suite 14.1;

2.2 Проверка работоспособности в симуляторе ISim

2.3 Код программы на языке VHDL для описания параллельного регистра

2.4 Проверка работоспособности в симуляторе ISim

2.5 Трансляция разработанного проекта, программирование ПЛИС.

3. Выводы.

Теоретическая часть

Сдвиговые регистры выполняются на основе D-триггеров с динамическим синхровходом.

Применение последовательного кода связано с необходимостью передачи большого количества двоичной информации по ограниченному количеству соединительных линий. При параллельной передаче разрядов требуется большое количество соединительных проводников. Если двоичные разряды последовательно бит за битом передавать по одному проводнику, то можно значительно сократить количество соединительных линий на плате (и размеры корпусов микросхем).

Принципиальная схема сдвигового регистра, собранного на основе D-триггеров приведена на рис. 2.20.

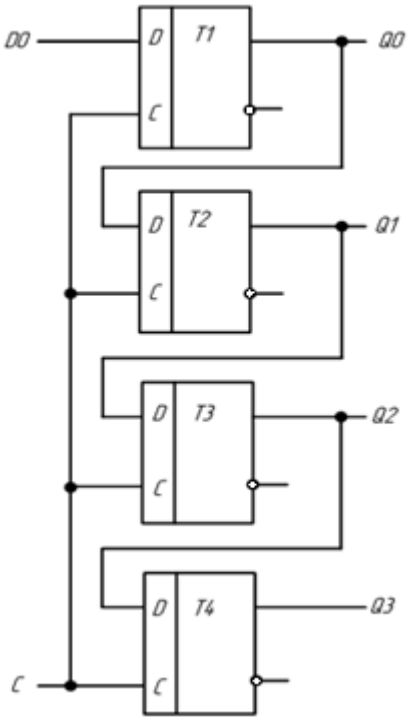


Рис. 2.20. Сдвиговый регистр на основе D-триггеров

Параллельные регистры осуществляют прием и выдачу информации в параллельном коде, а это значит, что для передачи каждого разряда используется отдельная линия.

Для записи информации в регистр на его входных выводах (**D0-D3**) нужно установить логические уровни, после чего на вход синхронизации (**C**) подать разрешающий импульс — логическую единицу. После этого на выходах **Q0-Q3** появится записанное слово. Регистры запоминают входные сигналы только в момент времени, определяемый сигналом синхронизации.

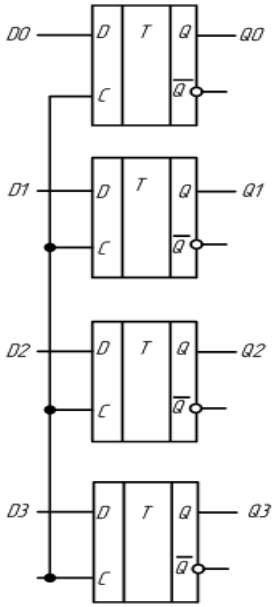


Рис. 2.21. Параллельный регистр на основе D-триггеров

Порядок выполнения работы

1. Разработка сдвигового регистра

1.1. Реализация сдвигового регистра в схемотехническом редакторе Xilinx ISE Design Suite 14.1

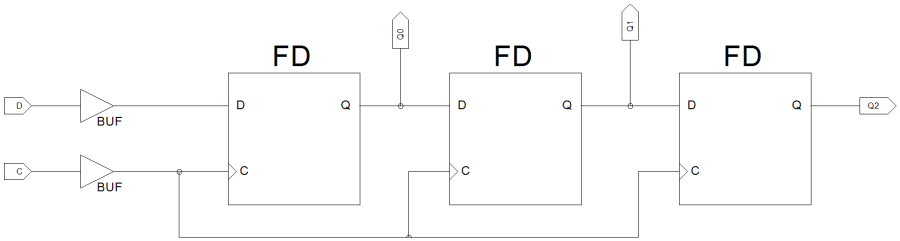


Рис. 2.22. Схема трехразрядного сдвигового регистра

1.2. Проверка работоспособности в симуляторе ISim

Составим на выходе код 101. Необходимо обеспечить сдвиг вправо, задавая необходимый сигнал на вход D, регистр срабатывает передним фронтом сигнала C.

№ состояния	Q_0	Q_1	Q_2
1	1	0	0
2	0	1	0
3	1	0	1

Приведем пример определения входных сигналов в TestBench.

```
   …
PROCESS
  BEGIN
D<='0';
C<='0';
wait for 50 ns;
```

```
C<='0';
wait for 100 ns;
D<='1';
C<='1';
wait for 50 ns;
C<='0';
wait for 50 ns;
D<='0';
C<='1';
wait for 50 ns;
C<='0';
wait for 50 ns;
D<='1';
C<='1';
wait for 50 ns;

…       //добавляем необходимые значения во временном
            промежутке с шагом 50 ns
WAIT;
 END PROCESS;
```

1.3. Код программы на языке VHDL для описания сдвигового регистра

library IEEE; use IEEE.STD_LOGIC_1164.ALL;	Подключаем все объявления пакета STD_LOGIC_1164, входящего в библиотеку IEEE
entity registr_posl is Port (clk : in STD_LOGIC; D : in STD_LOGIC; Q : out STD_LOGIC); end registr_posl;	Определяем сигналы, которыми объект будет обмениваться In – входные порты, A - двухразрядный вектор типа std_logic Out – выходные порты

architecture Behavioral of registr_posl is
Signal S : std_logic_vector(3 downto 0) := (others => '0');
begin
Process (clk)
Begin
If (rising_edge(clk)) then
S(3 downto 1) <= S(2 downto 0);
S(0) <= D;
End if;
End process;
Q <= S(3);
end Behavioral;
	if (условие) then
	// что делать, если условие верно
	else
	// что делать, если условие неверно
	Elsif – иначе если
	
	Срабатывания по фронту:
	rising_edge(clk) –вызов функции по переднему фронту
	falling_edge(clk) -- вызов функции по заднему фронту

1.4. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
process
  begin
D<='1';
wait for 100 ns;
D<='0';
wait for 100 ns;
D<='1';
wait for 100 ns;
    …
  end process;
```

Таблица 2.23

Индивидуальное задание

№	Индивидуальное задание	Запись числа
1	Трехразрядный регистр	001
2	Четырехразрядный регистр	0001
3	Трехразрядный регистр	010
4	Четырехразрядный регистр	0110
5	Трехразрядный регистр	100
6	Четырехразрядный регистр	0011
7	Трехразрядный регистр	101
8	Четырехразрядный регистр	0101
9	Трехразрядный регистр	110
10	Четырехразрядный регистр	1001
11	Трехразрядный регистр	011
12	Четырехразрядный регистр	1101
13	Трехразрядный регистр	110
14	Четырехразрядный регистр	0110
15	Трехразрядный регистр	101
16	Четырехразрядный регистр	0101
17	Трехразрядный регистр	011

2. Разработка параллельного регистра

2.1. Реализация параллельного регистра в схемотехническом редакторе Xilinx ISE Design Suite 14.1

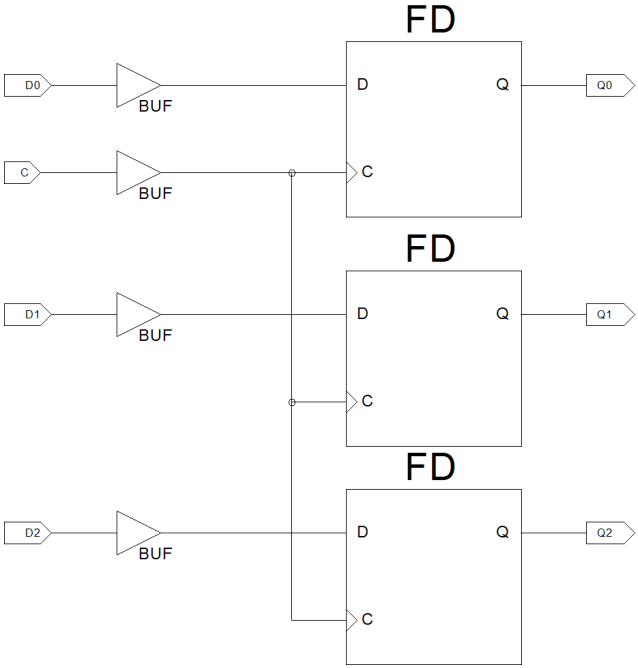


Рис. 2.23. Схема трехразрядного параллельного регистра

2.2. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
   …
PROCESS
  BEGIN
D0<='0';
D1<='1';
D2<='1';
C<='0';
wait for 50 ns;
```

```
D0<='0';
D1<='1';
D2<='1';
wait for 100 ns;
C<='0';
wait for 50 ns;
C<='1';
wait for 50 ns;

  …       //добавляем необходимые значения во временном
              промежутке с шагом 50 ns
  WAIT;
   END PROCESS;
```

2.3. Код программы на языке VHDL для описания параллельного регистра

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity reg is
  Port ( clk : in  STD_LOGIC;
         D : in  STD_LOGIC_vector(2 downto 0);
         Q : out  STD_LOGIC_vector(2 downto 0));
end reg;

architecture Behavioral of reg is
begin
Process (clk)
 Begin
  If (rising_edge(clk)) then
Q(2 downto 0) <= D(2 downto 0);
    End if;
 End process;

end Behavioral;
```

2.4. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
process
  begin
D<="011";
    wait for 100 ns;
    …
  end process;
```

Таблица 2.24

Индивидуальное задание

№	Индивидуальное задание	Запись числа
1	Трехразрядный регистр	001
2	Четырехразрядный регистр	0001
3	Трехразрядный регистр	010
4	Четырехразрядный регистр	0110
5	Трехразрядный регистр	100
6	Четырехразрядный регистр	0011
7	Трехразрядный регистр	101
8	Четырехразрядный регистр	0101
9	Трехразрядный регистр	110
10	Четырехразрядный регистр	1001
11	Трехразрядный регистр	011
12	Четырехразрядный регистр	1101
13	Трехразрядный регистр	110
14	Четырехразрядный регистр	0110
15	Трехразрядный регистр	101
16	Четырехразрядный регистр	0101
17	Трехразрядный регистр	011

Контрольные вопросы

1. Разработать схему сдвигающего вправо трехразрядного регистра на JK - триггерах.

2. Разработать схему сдвигающего влево трехразрядного регистра на D-триггерах.

3. Разработать схему сдвигающего вправо трехразрядного регистра на D- триггерах.

4. Разработать схему сдвигающего влево трехразрядного регистра на JK- триггерах.

5. Разработать схему реверсивного рехразрядного регистра на D- триггерах и ЛЭ “3И-НЕ”, "2И-2И-2ИЛИ-НЕ".

6. Разработать схему реверсивного трехразрядного регистра на JK -триггерах и ЛЭ “3И-НЕ”, "2И-2И-2ИЛИ-НЕ".

7. Разработать схему регистрового делителя частоты на JK - триггерах, К= 4.

8. Разработать схему регистрового делителя частоты на D - триггерах, К= 5.

9. Разработать схему регистрового делителя частоты на JK - триггерах, К= 6.

10. Разработать схему регистрового делителя частоты на D - триггерах, К= 7.

Лабораторная работа № 6

РАЗРАБОТКА И ИССЛЕДОВАНИЕ СЧЕТЧИКОВ

Цель работы

1. Получение практических навыков в разработке и исследовании счетчиков.

2. Привитие навыков проведения контроля работоспособности цифровых функциональных узлов комбинационного типа.

3. Приобретение практических навыков использования системы виртуального схемотехнического моделирования Xilinx ISE Design Suite 14.1.

Содержание работы

1. Разработка и исследование асинхронного вычитающего счетчика с последовательным переносом.

2. Разработка и исследование асинхронного вычитающего счетчика.

Литература

1. Бибило П.Н. Основы языка VHDL: Учебное пособие. Изд. 6-е. - М.: Книжный дом «ЛИБРОКОМ», 2014. – 328 с.

2. Бабак В. П., Корченко А. Г., Тимошенко Н. П., Филоненко С. Ф. VHDL. Справочное пособие по основам языка – М.: Издательский дом «Додэка-XXI», 2008. – 224 с.

3. Тарасов И.Е. Разработка цифровых устройств на основе ПЛИС Xilinx с применением языка VHDL. –Изд. 2-е. – М.: Горячая линия – Телеком, 2015. – 252 с.

Содержание отчета

Цель работы.

1. Разработка суммирующего счетчика;

1.1 Реализация суммирующего счетчика в схемотехническом редакторе Xilinx ISE Design Suite 14.1;

1.2 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

1.3 Код программы на языке VHDL для описания суммирующего счетчика;

1.4 Проверка работоспособности в симуляторе ISim с приведением временной диаграммы;

1.5 Трансляция разработанного проекта, программирование ПЛИС;

2. Выводы.

Теоретическая часть

Счётчик числа импульсов - устройство, на выходах которого получается двоичный (двоично-десятичный) код, определяемый числом поступивших импульсов. Счётчики могут строиться на двухступенчатых D-триггерах, T-триггерах и JK-триггерах.

Счетчики выполняют на запоминающих элементах – триггерах. Он фиксирует число импульсов, поступивших на его вход. В интервалах между ними счетчик хранит информацию об их числе. Совокупность единиц и нулей на выходах n триггеров (выходах счетчика) представляет собой n-разрядное двоичное число, однозначно определяющее количество прошедших на входе импульсов. Поэтому триггеры счетчика называют его разрядами.

Суммирующий счетчик увеличивает свое содержимое на единицу с поступлением каждого входного (счетного) импульса. Вычитающий счетчик аналогично уменьшает свое содержимое на единицу.

Комбинацией суммирующего и вычитающего счетчиков является реверсивный счетчик. У него может быть два входа, на один из которых поступают импульсы, увеличивающие его со-

держимое (суммирующие импульсы), на другой – вычитающие. Реверсивный счетчик может иметь один вход для суммирующих и вычитающих импульсов, а переключение с одного режима на другой осуществляется в нем сигналом на специальном входе.

В счетчик может предварительно заноситься число, для чего он имеет специальные входы. Каждый разряд счетчика может находиться в двух состояниях. Число устойчивых состояний, которое может принимать данный счетчик, называют его емкостью, модулем счета или коэффициентом пересчета.

Одним из основных параметров счетчика является его быстродействие. Оно оценивается минимальным интервалом между двумя соседними импульсами, с поступлением каждого из которых счетчик способен изменить свое содержимое. Счетчик является атрибутом многих цифровых устройств различного назначения. Его можно использовать по прямому назначению – для счета поступающих на его вход импульсов и для деления их частоты следования.

Триггеры счетчика соединяются между собой таким образом, чтобы каждому числу поступивших импульсов соответствовали единичные состояния определенных триггеров. Счетчик, у которого при поступлении входного импульса переключающий перепад передается от предыдущего триггера к последующему, называется счетчиком с последовательным переносом, а когда переключающий перепад на все разряды поступает одновременно (или почти одновременно) – счетчиком с параллельным переносом. Счетчики могут выполняться только на счетных триггерах. О состоянии разряда счетчика судят по потенциалу на прямом выходе триггера.

В большинстве случаев счетчики строятся таким образом, чтобы записываемое в них число было выражено в натуральном двоичном коде. В таком коде «вес» 1 на прямом выходе младшего разряда равен 1, а в каждом последующем разряде вдвое больше, чем в предыдущем.

Порядок выполнения работы

1. Разработка суммирующего счетчика
1.1. Реализация суммирующего счетчика в схемотехническом редакторе Xilinx ISE Design Suite 14.1

Построим схему суммирующего счетчика на JK-триггерах. Для этого на информационные входы необходимо задать режим переключения (J = K= 1). Счетные импульсы нужно подавать на вход триггера первого (младшего) разряда, каждым из которых он должен переключаться. Счетный вход очередного триггера должен подключаться к инверсному выходу предыдущего. В суммирующем счетчике состояние счетчика (двоичный код на его выходах) с каждым импульсом увеличивается на единицу.

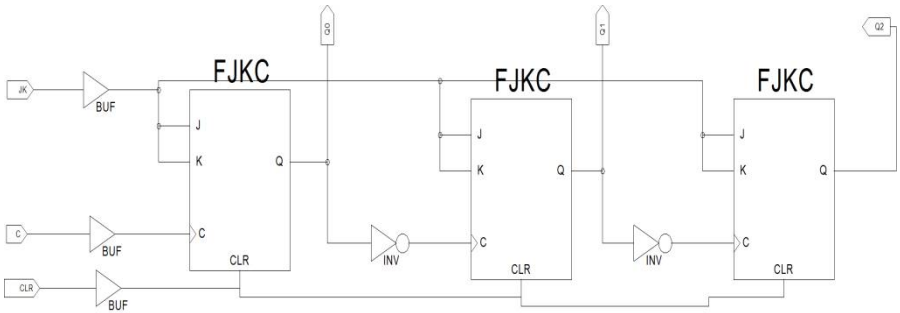


Рис. 2.24. Схема суммирующего счетчика

1.2 Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
   …
PROCESS
  BEGIN
BEGIN
JK<='1';
CLR<='1';
C<='0';
```

```
wait for 100 ns;
 …        //добавляем необходимые значения во временном
              промежутке с шагом 100 ns
 WAIT;
 END PROCESS;
```

1.3. Код программы на языке VHDL для описания суммирующего счетчика

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_unsigned.all;
 	
entity counter is
PORT(Clk, ena:IN STD_LOGIC;
Qa:OUT STD_LOGIC_VECTOR(2 downto 0));
end counter;
 	
architecture Behavioral of counter is
SIGNAL cnt:
STD_LOGIC_VECTOR(2 downto 0):="000";
 	
Begin
PROCESS (clk)
BEGIN
IF(clk'EVENT AND clk='1') THEN
if ena='1' then cnt<=cnt+1;
end if;
END IF;
END PROCESS;
Qa<=cnt;
end Behavioral;
	Подключаем все объявления пакета STD_LOGIC_1164, входящего в библиотеку IEEE
	Определяем сигналы, которыми объект будет обмениваться
	
	if (условие) then
	// что делать, если условие верно
	else
	// что делать, если условие неверно
	Срабатывания по фронту:
	rising_edge(clk) –вызов функции по переднему фронту
	falling_edge(clk) -- вызов функции по заднему фронту

В рассматриваемом примере приведен синхронный трехразрядный суммирующий счетчик, тактируемого фронтом синхроимпульса clk. Счетчик имеет цепь разрешения счета (синхронный сигнал ena). Активным считается сигнал ena высокого уровня.

1.4. Проверка работоспособности в симуляторе ISim

Приведем пример определения входных сигналов в TestBench.

```
constant Clk_period : time := 100 ns;
…
PROCESS
begin
ena<='0';
wait for 300 ns;
ena<='1';
…
process
  begin
ena<='0';
wait for 300 ns;
ena<='1';
…
     wait for Clk_period*10;
     wait;
  end process;
…
```

Таблица 2.25

Индивидуальное задание

№	Индивидуальное задание
1	Трехразрядный вычитающий счетчик на JK триггере
2	Четырехразрядный суммирующий счетчик на JK триггере
3	Четырехразрядный вычитающий счетчик на JK триггере
4	Трехразрядный вычитающий счетчик на D триггере
5	Четырехразрядный суммирующий счетчик на D триггере
6	Четырехразрядный вычитающий счетчик на D триггере
7	Трехразрядный вычитающий счетчик на JK триггере с предустановкой
8	Четырехразрядный суммирующий счетчик на JK триггере с предустановкой
9	Четырехразрядный вычитающий счетчик на JK триггере с предустановкой
10	Трехразрядный вычитающий счетчик на D триггере с предустановкой
11	Четырехразрядный суммирующий счетчик на D триггере с предустановкой
12	Четырехразрядный вычитающий счетчик на D триггере с предустановкой
13	Пятиразрядный суммирующий счетчик на JK триггере
14	Пятиразрядный вычитающий счетчик на JK триггере
15	Пятиразрядный вычитающий счетчик на D триггере
16	Пятиразрядный суммирующий счетчик на D триггере
17	Шестиразрядный суммирующий счетчик на JK триггере

Контрольные вопросы

1. Принцип построения и функционирования суммирующих счетчиков.
2. Принцип построения и функционирования вычитающих счетчиков.
3. Принцип построения и функционирования реверсивных счетчиков
4. Пересчетчики с исключением старших состояний: схема, функционирование.
5. Пересчетчики с исключением младших состояний: схема, функционирование.
6. Построить схему пересчетного устройства с исключением младших состояний на D-триггерах с прямым входом синхронизации, К=8.
7. Построить схему пересчетного устройства с исключением младших состояний на D-триггерах с прямым входом синхронизации, К=10.
8. Построить схему пересчетного устройства с исключением младших состояний на D-триггерах с прямым входом синхронизации, К=6.

ЗАКЛЮЧЕНИЕ

В учебном пособии излагаются основные сведения о порядке работы с версией пакета ISE Webpack.

На основе примера проекта простейших цифровых устройств продемонстрированы основные процедуры проектирования, чтобы максимально сократить промежуток времени между первым знакомством со средой проектирования ISE и конфигурированием проекта своего оригинального устройства в ПЛИС.

Рассмотрены синтез логической схемы по заданной таблице истинности, разработка триггеров, шифратора и дешифратора, мультиплексора и демультиплексора, регистров, счетчиков. Для каждой лабораторной работы приведены индивидуальные задания, контрольные вопросы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бибило П.Н. Основы языка VHDL: учеб. пособие. – Изд. 6-е. - М.: ЛИБРОКОМ, 2014. – 328 с.

2. VHDL. Справочное пособие по основам языка / В.П. Бабак, А.Г. Корченко, Н.П. Тимошенко, С.Ф. Филоненко. – М.: Додэка-XXI, 2008. – 224 с.

3. Тарасов И.Е. Разработка цифровых устройств на основе ПЛИС Xilinx с применением языка VHDL / И.Е. Тарасов. – Изд. 2-е. – М.: Горячая линия – Телеком, 2015. – 252 с.

ОГЛАВЛЕНИЕ

Учебное издание

Строгонов Андрей Владимирович
Кошелева Наталья Николаевна
Буслаев Алексей Борисович

ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ В БАЗИСЕ ПЛИС: ЛАБОРАТОРНЫЙ ПРАКТИКУМ

В авторской редакции

Подписано к изданию 28.09.2017.
Объем данных 3,7 Мб.

ФГБОУ ВО «Воронежский государственный технический университет»
394026 Воронеж, Московский просп., 14